PERCU: A Holistic Method for Evaluating High Performance Computing Systems

William TC Kramer

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-143
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-143.html

November 5, 2008

Acknowledgement

See text of report

PERCU: A Holistic Method for Evaluating High Performance Computing Systems

by

William T.C. Kramer

B.S. (Purdue University) 1975
M.S. (Purdue University) 1976
M.E. (University of Delaware) 1986

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James Demmel, Chair
Professor David Culler
Professor James Siegrist

Fall 2008

The dissertation of William T. C. Kramer is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

Fall 2008

PERCU: A Holistic Method for Evaluating High Performance Computing Systems

by

William T.C. Kramer

Abstract

PERCU: A Holistic Method for Evaluating High Performance Computing Systems

by

William T.C. Kramer

Doctor of Philosophy in
Computer Sciences

University of California, Berkeley

Professor James Demmel, Chair
Professor David Culler
Professor James Siegrist

PERCU is a comprehensive evaluation methodology for large-scale systems that expands Performance analysis to include Effective work dispatching, Reliability, Consistency, and Usability. The PERCU approach and its components can be used for initial system assessment as well as for on-going quality assurance of High Performance Computing (HPC) and other systems. PERCU leverages work that has to be done in traditional benchmarking and acquisition approaches by compositing existing data to gain additional insights.

A key contribution is the Sustained System Performance (SSP) concept which uses time-to-solution for assessing the productive work potential of

systems for an arbitrary set of applications. The SSP provides a fair way to compare systems deployed at different times and provides a method to assess sustained price performance in a comprehensive manner. This work also discusses the Effective System Performance (ESP) test, developed to encourage and assess improved job launching and resource management – both important aspects for a productive HPC system. Reliability is the third characteristic of a productive system. This work explores the major causes of failure for very large systems and suggests improved methods for *a priori* assessment of the reliability of HPC systems. Consistent execution of programs is a metric often overlooked in assessments, but is a key service quality feature. This work shows how lack of consistency impacts quality of service and defines approaches for assessing and improving consistency. Usability is discussed for completeness and as future work.

PERCU can be used, in all or part, and with a limitless scale of detail and effort. At its simplest, it is a framework for holistic evaluation. In its detail, it introduces a set of methods for measurement of key parameters that impact quality of service on HPC systems. The use and impact of each PERCU element is documented for multiple systems, mostly using systems evaluated at the National Energy Research Scientific Computing (NERSC) Facility.

_____

Professor James Demmel, Chair

Dedication

This work is dedicated to the two ladies in my life that give me the inspiration to and the joy of excellence. My daughter Victoria, who works harder than anyone I have ever met, from her first days, has a love of learning, a sense of humor and a style that inspire me. My wife Laura is my complete partner and friend, for infinity. She inspires me with her continuous evolution and re-invention of herself, her unselfishness and her intelligence that are beyond anyone I have ever met.

To these two outstanding ladies, I dedicate this work and my life as a small token of my thanks and love.

Table Of Contents

List of Figures

List of Tables

List of Equations

Acknowledgements

Many people who work very hard to assure HPC systems work well and support the people who use HPC to design new things and craft new understandings have inspired this work. I want to thank each and every person for their contributions in this exciting world of helping HPC make fantastic impacts on the lives of millions and billions of people. While I am not able to name everyone in this section, I thank them all for their enthusiasm and technical leadership. This work was supported by the Office of Computational and Technology Research, Division of Mathematical, Information and Computational Sciences of the U.S. Department of Energy, under contract DE-AC03-76SF00098.

This work coincides with related efforts at NERSC (National Energy Research Scientific Computing Facility) at Lawrence Berkeley National Laboratory (LBNL) to improve the impact large system for scientific computing. I thank all the people at NERSC who have been involved with fielding and supporting the systems and applications. I also thank the computational research community that is discussed in this document. I could not have accomplished this work without the enthusiastic help and support of the entire NERSC staff. They are my cheerleaders and my consultants. The people at NERSC made PERCU real instead of an academic concept. It is a complement to have people using the ideas held in this document to solve real problems. The impact the NERSC staff has on science and progress is

beyond all reason. I am in awe of them and grateful for our time together. It has been an honor and inspiration to work with them to make NERSC, as one of our users wrote, “absolutely the best Supercomputing Center in the known universe”. Unless otherwise noted, all the people noted below are associated with NERSC or LBNL.

I would like to thank the people who have made the SSP concept successful, including Ms. Katie Antypas, Dr. David Bailey, Dr. Jonathan Carter, Dr. Helen He, Dr. Erich Strohmaier, Dr. David Skinner, Mr. John Shalf, Dr. Harvey Wasserman, Dr. Richard Gerber, Dr. Lenoid Oliker, Dr. Adrian Wong and Ms. Lynn Rippe. Several people participated in the efforts to define, create, refine and use the Effective System Performance Test. The concept of the need for a test to reflect a *“Day in the life”* of an HPC system came during conversations with Dr. David Bailey and Mr. Michael Hennessey of IBM. The author of this document defined and wrote the first ESP methodology later that evening which lasted until just before the sun came up. ESP-1 was co-created with Dr. Adrian Wang, Dr. Lenoid Oliker, Ms. Theresa Kaltz and Dr. David Bailey and was used to assess NERSC-3. Dr. Adrian Wang did the majority of the coding work for ESP-2. The use of ESP-2 to evaluate job schedulers on the same platform was conceived by and led by the author. Mr. Tom Davis and Mr. Jason Gabler did the ESP-2 configuration, system administration and batch configuration definition for each test. Finally, the ESP experiments for the NERSC-5 system were carried out by Dr. Joe Glenski and his team at Cray.

Reliability work overlaps with work done for the Parallel Data Storage Institute, a DOE SciDAC project, which partially funded the work to correlate the NERSC system reliability data. As the NERSC Principle Investigator for this work, I appreciate the efforts of my NERSC coworkers Mr. Akbar Mokhtarani and Mr. Jason Hick as well as the on-going efforts of the NERSC staff to properly track the system data.

Determining system consistency evolved to be an important aspect of system evaluation and operation. The work was originally motivated to provide a guideline for fielding the large IBM SP system called Seaborg. I worked with Mr. Clint Smith, a fellow graduate student at UC Berkeley, on the first LBNL technical report and first paper on consistency, both of which are cited in the references for Chapter 7. This work evaluated the consistency of multiple MPP systems. The second paper was coauthored with Dr. David Skinner and looked at the facets that contributed to consistency or its inverse, variation. Dr. Richard Gerber followed the system test methodology defined here and determined and investigated the causes of inconsistency on the Power-5 system, Bassi. It is also important to acknowledge excellent work by the IBM and NERSC systems staff, including Mr. Nick Cardo, Ms. Tina Butler and Mr. Scott Burrrow (IBM), who actually identified enough evidence for the root causes to be determined to correct several issues within the IBM systems. Without finding the root causes of inconsistency, it would never have been possible to know that consistent HPC systems are in fact achievable.

I would also like to thank the organizations that submitted Acquisition and Requirements documents for summarization in this effort, including the National Center for Atmospheric Research, Lawrence Livermore National Laboratory, the DOD HPC Modernization Office, the Canadian Meteorological Center, the European Center for Medium Range Forecasting and others that asked not be mentioned. Thank you as well to the Pittsburgh Supercomputer Center and the National Energy Research Scientific Computing Center for access to systems that were used to develop and test these concepts.

This work would not have been possible without the participation of many vendors, who not only provided data from their systems, but also spent significant staff resources correcting issues and improving systems to respond to the observations of the PERCU evaluations. The list of vendors includes International Business Machine Corporation, Cray Inc., Linux Networx, Compaq Computer (now part of HP), Silicon Graphics Inc., Platform Computing, Cluster Resources, Inc., and several others.

There are several people within the Office of Advanced Scientific Computing and Research Office of the DOE Office of Science that have been very supportive of this work, including Dr. Daniel Hitchcock, Dr. Walter Polansky, Dr. Buff Miner, Dr. Thomas Kitchens, Dr. Fred Johnson, Dr. Barbara Helland, Dr. Yukiko Sekine, Dr. Michael Strayer and Dr. Raymond Orbach. Many of my colleagues encouraged me in my quest for a PhD, not only helping me evolve my thinking but also keeping my spirits up. In

particular, Dr. Frank William and Dr. Ron Bailey were true friends with their encouragement and confidence. I would like to acknowledge the support Dr. Horst Simon gave this work. I want to thank Mr. James Craw, my long time colleague, whose group ran many of the systems in this analysis and often carried out the improvements that were identified. Mr. Howard Walter and Ms. Francesca Verdier responded to the information identified in this report and helped support the effort to make the improvements in NERSC systems that are documented. In particular, I owe a debt of gratitude to Dr. William McCurdy and Dr. Charles Shank who whole-heartedly encouraged this endeavor and have been role models.

The staff and faculty in the Computer Science Division at the University of California at Berkeley have taught me computer science for the 21st century. The entire department took a chance that a part-time graduate student would be able to accomplish the program and add to the department. I have learned in every class I took, but more importantly, I have learned how to approach researching problems. In particular, I am deeply thankful for the on-going support of Dr. James Demmel – who steered me the through the process and was extremely patient with the distraction of my real job. Dr. David Culler not only introduced me to an entirely new arena of large scale systems with self configuring networks, but also helped simplify the analysis and writing of this document. Dr. Katherine Yelick and Dr. Susan Graham were supportive throughout my time at UCB. Not only was Dr. David Patterson willing to chair

my qualifying committee, but I also learned from him how to learn an entirely new topic simply by asking very good questions and keeping an open mind.

The NERSC scientific user community is responsible for motivating this work. The work they do benefits the entire country and it is a pleasure to develop ways to make their life easier.

Most important and finally, I am infinitely grateful to my family. My wife, Laura Kramer, did everything possible to provide the opportunity to pursue making this dream. She took on many things that I could have been doing in order to give me the time to do this work. She helped complete this work by reading and correcting every word and most importantly using her wonderful project management skills to assure this effort completed. Together, Laura and I got our first Computer Science degrees at Purdue University. My daughter Victoria is an inspiration with her love of learning and adventure, her persistence and her wonderful hugs. I look forward to now spending a more time with them.

Chapter 1: Introduction and Motivation

1.1 Chapter Summary

The Performance, Effectiveness, Reliability, Consistency and Usability (PERCU) method is a holistic*, user based methodology for evaluating computing systems, which, in this work, is applied to high performance computing (HPC) systems. It enables organizations to use flexible metrics to assess the performance of HPC systems and continually monitor them against the requirements and expectations. PERCU expands Performance analysis to include Effective work dispatching, Reliability, Consistency and Usability. The PERCU approach and its component's framework can be used for initial system assessment as well as on-going quality assurance of HPC systems.

The key contribution of this work to performance evaluation is the Sustained System Performance (SSP) concept which uses time-to-solution to assess the productive work potential of systems for an arbitrary large set of applications. The SSP provides a way to fairly compare systems which may be introduced with different time frames and also provides an exact method to assess sustained price performance.

* The Merriam-Webster on-line dictionary defines holistic to be "elating to or concerned with wholes or with complete systems rather than with the analysis of, treatment of, or dissection into parts." - http://www.merriam-webster.com/dictionary/holistic. This is an appropriate description of PERCU looking at the complete system.

This work also introduces the Effective System Performance (ESP) test that is developed to encourage and assess improvements for job launch and resource management features of systems – both important aspects for productive computing systems. Reliability is the third characteristic of a productive system. This work explores the major causes of failure for large systems and suggests improved methods for *a priori* assessment of the potential reliability of HPC systems. Consistent execution of programs is a metric that is often overlooked in system assessments, but a key quality of service that will be missed if it is not present. This work provides background for why consistency can impact quality of service, what causes inconsistency, and it defines approaches to assessing it. Usability for HPC systems, itself a possible topic of an entire dissertation, is discussed for completeness and as future work.

Another goal of PERCU is to utilize the efforts that are typically done for system assessment, such as running benchmark tests, and not craft entirely new methods, unless new methods are needed to fill in gaps in the holistic evaluation approach (e.g. ESP is a new test). Hence, several of the PERCU techniques leverage typical tests in order to either provide new insights or to cover gaps. Other aspects of PERCU create entirely new approaches to assessing systems. The result is the flexibility for organizations and individuals to apply PERCU in a way that is compatible with their existing practices, while at the same time, gaining improved insights and new capabilities.

PERCU is used, in all or part, and with a wide range detail and effort. At its simplest, it provides a framework to consider holistic evaluation of large systems. In its detail, it introduces a set of new measurement methods. The use and impact of each component is documented for multiple systems, using mostly systems that have been evaluated at the National Energy Research Scientific Computing (NERSC) Facility. The impact of this work is evident in the fact several organizations adopted the use of PERCU concepts in their own operations.

1.2 Steps Taken To Develop PERCU Methodology

The approach taken to develop the PERCU methodology consisted of the following steps.

1. Accumulating an inventory and assessing approaches to system evaluation used by major HPC organizations, users, system managers, researchers and evaluators. It involves review of the evaluations for a number of systems. This identifies common activities that many parties carry out, e. g. what is being done that works and what are the problem or the gap areas.
2. The observations of the important system characteristics and approaches to assess them have been validated with the community at technical conferences and workshops.

3. Likewise, the author had detailed discussion of proposed methods with vendors, sites, and users to determine improvements and effectiveness.
4. The end result, the PERCU method of holistic evaluation, defines frameworks that are used at appropriate scale and complexity for HPC system assessments.
5. For each area of PERCU, new assessment frameworks, such as the SSP and ESP tests, are now implemented and one or more examples of each framework's use completed and evaluated on real systems in real environments as a proof of the PERCU concepts.
6. The PERCU Method and associated frameworks were evaluated and fine-tuned over time for multiple systems. The experiences using PERCU are recorded with observations for improvement. Where feasible, iterative changes to the methods improve the frameworks.
7. Finally, the frameworks, and indeed the PERCU methodology, are made available to other organizations and individuals for their usage.

1.3 PERCU's Importance to the HPC Community

Current methods of evaluating HPC systems are incomplete, disjoint, and insufficient for future highly parallel systems. In June 2008, Dr. William Camp of Intel (Camp 2008) presented estimates that between 2003 and 2012,

$107+ billion dollars will be spent on HPC systems. During this time period, the HPC market is growing at twice the rate of all other computing areas. 20 – 25% of Intel server CPU chips are being sold for what Intel defines as HPC/High End technical markets. On the other hand, vendors such as Cray and IBM estimate they spend 10's of millions of dollars responding to purchase requests, all of which are unique and are based on the site's interests. Making the evaluation of HPC systems more efficient and more accurate will provide benefits purchasing organizations, vendors, and users alike.

Numerous studies (Graham, Snir and Patterson 2004), (Holbrock and Shaw 2005) indicate the role HPC systems play in science, safety, and commercial competitiveness is significant and growing. The investments organizations make in high performance computing is strategic and, in many areas, critical to the organization's long term success. Hence the efficient evaluation of technology allows the timely and cost effective selection of systems which have the potential to enhance user productivity.

This work addresses the challenges of efficiently and thoroughly evaluating large scale technology from a holistic, user point of view. It furthermore introduces new methods that are important in order to assure systems provide high quality of service.

1.4 Organization

This document is organized into chapters each of which describes one facet of the PERCU approach. The chapters provide background on the issues in the area and propose new methods to add to the traditional methods. In each area and in the appendices, there are discussions of how the method is applied at the NERSC Facility and what beneficial outcomes resulted. The NERSC status shows real world use of PERCU and the benefits that contribute to user productivity. The rest of this section includes a brief summary of each chapter.

1.4.1 Introduction and Motivation

Chapter 1 briefly discusses some of the motivation and the organization for the document. The motivation is four fold.

- First, HPC systems play an increasingly strategic and irreplaceable role in many endeavors – Government, Academic and Industrial – and a method accurately assessing the best solutions for new technology is important to all parties.

- Second, HPC systems are extremely complex and simplified, low level; point tests do not capture the complex and subtle interactions that greatly influence the ability of HPC systems to meet expectations.

- Third, the HPC area is growing at twice the rate of other IT sectors, and providing good metrics and guidance to system creators will improve the productivity and cost effectiveness of the systems.

- Finally, it is critical to use evaluation methods that take into account all the system characteristics that HPC users need to be successful.

1.4.2 Comparing Evaluation Requirements

The community that assesses large scale computational resources often focuses solely on performance as the way to measure a computing system. As seen with the PERCU methodology, there are other key factors in evaluating systems. Chapter 2 develops a set of criteria sites and their user communities want in HPC systems. The criteria are based on analysis of HPC sites' acquisition documents and other factors. While the style and breakdown of features vary, there was commonality in the categories of attributes organizations request for their systems. The major categories are below.

- Performance, which is essentially how fast a system processes work if everything is working extremely well.
- Effectiveness in scheduling and launching work, which addresses the likelihood users can get the system to do their work when they need it.
- Reliability, which is the likelihood the system is available to do work and operates correctly.

- Consistency, which is how often the system will process the same or similar work correctly and in approximately the same time duration.
- Usability, which is how easy is it for users to get the system to process their work as fast as possible.

Usability and Performance are categories that had the most number of requirements but combined represent only half the factors used for system purchasing decisions. The performance factors represent 22% of the explicit factors. Consistency was an area that is a relatively recent concern and not as often addressed in older proposal requests. While the distribution of factors is important to determine the areas that are needed to holistically assess a system, the distribution does not imply the relative influence on the ultimate evaluation or purchase decisions. Nonetheless, categorization of system attributes addresses the entire system and can be viewed as a holistic description of the system attributes needed to provide a productive, high performance computing system.

When organizations write the Requests for Proposals (RFPs) for the purchase of a new HPC system, the requirements can be classified in a consistent manner into the five technical categories which form the PERCU methodology: Performance, Effectiveness, Reliability, Consistency and Usability. The analysis was done of a number of actual, real RFP's, and it was found that 84% of the requirements, and almost all the technical factors, fit into these categories. The remaining 16% were non-technical factors, such as requirements for security clearances, management meetings, proposal

guidance and other provisions that do not relate to the technical system being evaluated.

1.4.3 Sustained System Performance Method

Chapter 3 discusses the performance aspects of PERCU by introducing SSP, which is a framework that enables evaluation of a system's performance using time-to-solution while at the same time accommodating any number of application areas. It defines the equations for SSP and provides a theoretical basis for the framework. It uses simple examples to provide the motivation for use and implementation of SSP. This section describes how SSP enables time-to-solution for different application domains to be compared across systems to determine cost performance and value.

There are multiple goals for the design of the SSP method that were achieved. The methodology should be flexible so it applies to different:

- System use cases;
- Workloads and usages;
- System scales (system size, cost, scale, etc.) and flexible degrees of effort to do evaluations from "quick and dirty" to highly formalized;
- Levels of Quality of Service (duty cycles, reliability, etc.) and
- User communities.

Furthermore, the methodology should efficiently serve four purposes:

1. Differentiate (*select*) a system from among its competitors;

2. Validate the system works the way expected once a system is built and/or is delivered;

3. Assures systems perform continues as expected throughout its lifetime; and

4. Guide future system designs and implementation.

Another consideration in designing the SSP was to add as little additional work to traditional system evaluation methods as possible. SSP uses the benchmarking that most organizations already do and improves insight as a composite measure. These goals do not just apply to SSP, but can be seen in the other aspects that make up PERCU.

1.4.4 Practical Use of SSP for HPC Systems

Chapter 4 describes using SSP to assess performance of large systems. It includes analysis of actual uses of SSP over a 10 year period to evaluate the performance and price performance of five HPC systems at NERSC. The latest version of the SSP suite is available at:

- http://www.nersc.gov/projects/procurements/NERSC6 and

- http://www.nersc.gov/projects/ssp.php.

1.4.5 Effectiveness of Resource Use and Work Scheduling

Effectiveness is a component of the PERCU method that assesses the ability of a system to efficiently provide users access to the performance and capabilities in the system. To measure effectiveness, a system utilization benchmark, the Effectiveness System Performance (ESP) test, is developed as part of PERCU. Chapter 5 provides a brief description of how ESP evolved along with the design goals for the test. Also discussed is the impact of using ESP in different areas. The chapter summarizes (and Appendices E and F provide much detail) using ESP to evaluate different job scheduling software packages. ESP-2 is packaged in a freely available software archive, with facilities for simple installation and execution. It is located at http://www.nersc.gov/projects/esp.php.

1.4.6 Reliability

Reliability is the next aspect of system productivity. It is challenging, yet critical to proactively assess reliability of a system before it is purchased. Chapter 6 studies failure causes spanning six major HPC systems over five years. It identifies the major reasons HPC systems fail. It shows that, at least for the systems included in the study, system wide outages were more often caused by software than hardware. The chapter discusses the reasons individual jobs fail on one system and discusses improvements that resulted from that analysis. The chapter suggests ways to improve the up-front

assessment for systems from the reliability point of view as well. Much of the reliability data discussed is available at http://pdsi.nersc.gov.

1.4.7 Consistency of Performance

Systems can support 10-20% more work after consistency issues were addressed, as shown in several NERSC systems. It is shown that very large systems can be made consistent. The loss of cycles due to inconsistency is avoidable for properly configured, designed and managed systems to the degree that inconsistency can be less than a few percent.

Chapter 7 discusses the Coefficient of Variation (CoV) composite metric that is used with a variety of benchmark testing to assess consistency. CoV and other approaches provide measurements that led to resolving issues causing inconsistency. The chapter includes the results of a study on real production systems related to consistency and improvements to the systems based on the metrics.

1.4.8 Usability – Something for the Future

Scientists want to know how much harder it is to use HPC systems than their standard platforms and tools. They want to know how much more effort is required to get a certain amount of work done on the HPC system rather than their desktop systems. Chapter 8 surveys the overall area of usability research and comments on what might be useful for HPC projects.

1.4.9 PERCU's Impacts, Conclusions and Observations

The PERCU method has seen positive impacts for sites using it and its associated components. In fact, the evolution of the method helped at least one HPC site to be called the "best run centralized computer center on the planet" by one of its major users (NERSC User Survey 2003). Chapter 9 summarizes some of the impacts and ways PERCU has been employed to assist in selection and monitoring HPC systems.

There is much more to be done in each of the areas of PERCU; Performance, Effectiveness, Reliability, Consistency, and Usability. PERCU is being used by NERSC. Other sites are using some of the frameworks and components. The methodology will help organizations get better performance, have more effective systems, give users a more reliable system, and be able to measure consistency. The chapter also lists some ideas for further study.

The world of HPC is expanding daily. It is my sincere hope that the work presented here contributes to the effort to get systems that can better solve the world's complex problems.

Chapter 2: Comparing Evaluation Requirements

2.1 Chapter Summary

The evaluation factors used by a wide spectrum of organizations to assess computer systems can be classified into a small number of categories in a consistent manner, regardless of the size and scope of the system being evaluated. Technical classification into five categories of Performance, Effectiveness, Reliability, Consistency and Usability (PERCU) account for 84% of the factors used and almost all the technical factors. The remaining 16% were either not technical factors, such as requirements for security clearances, management meetings, etc.

Part of the motivation for developing the PERCU method was the realization that the majority of the factors in purchasing decisions are not related to performance. This raised the question of what other factors are used and how can they be assessed. Further, the performance area had potential for improvement as well, so there could be a more consistent approach to understanding the potential for different systems to do different amounts of work.

The factors used to evaluate systems range from the very specific to extremely general provisions. Evaluating systems is done in many contexts, from very focused evaluations for single low level features to broad evaluations of entire systems. The types of factors used in the HPC community for evaluation point to the issues and features of systems that are

of the most concern. The question is whether different evaluators use similar factors when assessing systems and, if so, can some of the categories be made more consistent. There are many examples of performance being an important factor in the academic literature, indeed there are entire conferences devoted to performance evaluation. However, as the analysis below shows, evaluation factors for performance features of a system represent 22% of the factors used in purchasing decisions. While the distribution of factors is important to determine the areas that are needed to holistically assess a system, the distribution does not imply the relative influence of factors on the ultimate evaluation decision. For example, it may be that the performance factors play a relatively large role (have more weight) in the final purchase decision.

But is evaluating performance sufficient to understand how well a computing system will meet the needs of its client community? The simple answer is performance alone is not sufficient – in fact there are five categories of factors that are used to evaluate systems. One way to investigate this question is to look at the evaluation factors used for system purchases. In some ways, the factors used in purchases may be a better indication of what attributes are most important because a) the originators of the factors will pay real money for the systems they evaluate and b) the clients of the systems will be more or less productive based on how well the entire system performs for their purposes. This section looks at factors used in the acquisition of HPC systems ranging from $3M to $200M and

categorizes the requirements organizations are using to evaluate and buy HPC systems.

Acquisition methods vary a great deal based on the size of the investment, the purpose of the system, the mission of the organization and the state of technology[*]. Acquisition requests are called different names, including Request for Proposal (RFP), Request for Bid (RFB), Request for Quotation (RFQ), Tenders (a common European term) and Solicitations, to name a few. For the sake of simplicity, in this work, we use the common United States term, Request for Proposal (RFP) to discuss any of the methods used to assess and acquire systems since the methods differences are more based in acquisition rules and legal regulations of the governing policy than they are a differences for expressing to potential bidders what is required and what will be evaluated.

While each RFP is unique, there are similarities across RFPs. For the most part, organizations consider their RFPs sensitive and do not release them, but we were able to collect RFPs from different organizations, both within the United States and from Europe.

2.2 Analysis Method

Assessment of purchase decision evaluation factors, often called requirements, in acquisition documents is a subjective process since there is

[*] Peter Ungaro, currently President of Cray, Inc. and formerly responsible for HPC sales at IBM, estimated that Cray responded to approximately 100 HPC RFP's a year, split approximately equally between government and industry. In this case, HPC systems are defined as systems that cost over $1.5M.

no formal language for specification. In this classification, the first step was to group all the factors expressed in each RFP with other similar requirements in other RFPs. For example, *"1 GB of memory per compute processor"* is a factor that is essentially the same as *"Two (2) Gigabytes of memory per processor."* Both of these factors specify the amount of memory expected on a per processor basis in absolute terms, even if they are requesting different amounts of memory. However, not all comparisons are as clear cut. Take a factor such as *"Require minimum benchmark memory".* This is also a requirement of the amount of memory that is required of the system, but it is relative to the amount of memory necessary to execute the specific benchmarks that were used with that procurement. The evaluators decided to represent the amount of memory needed through its benchmarks, not as an absolute amount. This could be because the organization did not want to predefine the concurrency to solve the benchmark problem or the problem memory size could be adjusted to processor configurations. Regardless, the evaluation factors address how much memory the system has, and thus, are related.

Another similar requirement example includes "*IEEE 754 32 bit floating-point numbers"*, "*IEEE 754 64 bit floating-point numbers"*, "*IEEE Floating Point"* and "*IEEE 64 bit FP M29"* or in another instance, *"compilers and their related development environment, profiling and debugging tools: C (ISO/IEC 9899:1999) compiler, Fortran 90 (ISO/IEC 1539-1:1997) compiler. C++ (ISO/IEC 14882:2003) compiler"* and "*FORTRAN 90 and C compilers and*

libraries". These factors clearly indicate the evaluator is expecting compliance to defined standards. Factors that have less precise definition, but still with the same intention can be seen in the example of statements from two RFPs - *"consistent performance (within 10%) of dedicated applications regardless of which batch nodes is used. The Offeror should describe any attributes of the proposed system which would result in non-compliance"* and *"All nodes identical"*. Both of these evaluation factors are expecting similar performance across the system.

Most of the evaluation factors were standalone statements. However, some factors had sub-factors to make up a full requirement. These are classified as Major and Sub-major evaluation factors. The notation used when summarizing Major and Sub-major factors is N.S. where N is the number of major evaluation factors and S is the number of significant, but sub-major level evaluation factors. For example, a major factor may be *debugging cluster-wide applications*. The Sub-major factors for the major evaluation factor could be for *a visual representation* of the debugging information, being able to *set conditional breakpoints* or tools to assisting in *memory leak detection*.

By grouping similar factors as described above, patterns emerge. For example, most RFPs have evaluation factors that indicate the need for FORTRAN, C and C++ compilers. Since these RFPs are for HPC computational systems, other languages and compilers are less common.

RFPs are more or less specific on the details of the expected compiler functions and standards. Another grouping that emerges clearly is factors that specify the required, and/or desired, processing rate for computations – performance. Often this is expressed in some overall term as well as through the use of benchmark tests. Evaluation factors are also common for interconnect rates and input and output. There are common factors for quantities of things. For example, the amount of memory is one area that is often expressed as is the capacity of disk storage.

The entire set of factors and how they are categorized is posted at http://www.nersc.gov/~kramer/UCB/Dissertation/Data along with other data associated with this research.

2.3 Summary of Evaluation Factor Analysis

Evaluation factors in most acquisition documents are separated into two major categories; mandatory and desired. Different acquisition methods used different terms for each of these categories, such as *minimum*, *mandatory*, and *baseline* for the former, and *performance, non-mandatory*, and *desired* for the latter. There can be Major and Sub-major factors that are either mandatory or desired in any RFP.

Mandatory factors are those that, for the most part, have to be met by the proposed systems in order for it to be considered at all for purchase. The use of mandatory and desired factors varies dramatically by different organizations. One RFP had only one mandatory evaluation factor, which was

to meet a certain level of computational performance. On the other hand, another RFP had only mandatory factors and no desired factors. The RFPs that have more than one mandatory and more than one desired factor had an average of 15 desired factors for every one mandatory factor. The complete list of factors can be found at http://www.nersc.gov/~kramer/UCB/Dissertation/Data.

RFP / Type of Factor	RFP 1	RFP 2	RFP 3	RFP 4	RFP 5	RFP-6	RFP-7	RFP-8	RFP-9	RFP-10	RFP-11	RFP-12
Mandatory Major	17	79	92	18	15	12	10	30	1	39	83	208
Mandatory Sub-Major	0	1	0	0	0	0	0	9	0	6	5	118
Desired Major	35	8	2	17	24	25	18	0	116	7	192	20
Desired Sub-Major	0	0	0	0	4	2	0	0	10	1	0	0
Total Major	52	88	94	35	39	37	28	30	116	46	275	228
Total Sub-Major	0	1	0	0	4	2	0	9	10	5	5	128
Ratio of Desired Major to Mandatory Major	2.1	0.1	0.02	0.9	1.6	2.1	1.8	0.0	116.0	0.2	2.3	0.1
Ratio of Sub-Major to Major	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.30	0.00	0.11	0.02	0.51
Approximate Cost	$50M	$30M	$200M	$33M	$27M	$6M	$3.4M	$1.5M	$200-250M	$40-45M	$39.4	$37.3M

Table 2-1: Summary of Mandatory and Desired Evaluation Factors for 12 RFPs.

This ratio of desired to mandatory factors shown in Table 2-1 seems low given the trend in US federal acquisition over that past 15 years to move from proscribed specifics to more general provisions. On closer examination, Table 2-1: shows three RFPs that have ratios less than 0.1 - lower by an order of magnitude than the others. Without these three RFPs, the ratio is 1.6.

It is interesting to note that the two RFPs with a low ratio represent organizations that are heavily oriented to “production” computing for weather forecasting and engineering design. Because these “mission oriented” systems have well defined and limited scope workloads, as well as strict operational constraints, it may be the organizations doing the evaluation feel more pressure to specify a solution with which they are familiar.

There are two major contradictions to this observation. First, the US Advanced Strategic Computing Initiative (ASCI) is a very mission oriented program – using simulations to safe-guard nuclear weapons stock pile. The ratio for the RPF from one of the ASCI sites is well above the average, but the acquisition was done for systems during the development part of the program rather than the production part. On the other end of the spectrum, the two RFPs that support the general science community at US universities – a very broad workload – have a very low ratio of mandatory to desired factors.

There is a wide range of the number of total evaluation factors in these RFP’s, with the total number of evaluation factors ranging from 28 to 356. The range for the major evaluation factors is almost as large, from 28 to 275. There is little correlation between the expected cost of the system and the number of total factors, with a correlation coefficient of 0.117. There is a slightly higher, but still insignificant, correlation between the system cost and the number of mandatory factors – with a correlation coefficient of 0.181.

2.4 Overall Categories of Evaluation Factors

As indicated above, evaluation factors in one RFP could be compared to similar factors in other RFPs. Using this comparative process, RFP factors were grouped into categories, such as compilers, debuggers, interconnect, memory, reliability, service, support and other characteristics. Factors also dealt with:

- functionality (e.g. UNIX POSIX User Interfaces, parallel file systems, job management systems, standards); and
- rates of performance (e.g. floating point performance, bandwidth performance, latency performance); and
- capacity or amounts (e.g. amount of disk storage and number of connections).

Some factors that were highly specialized to a given acquisition (e.g. the need to have security clearances for staff supporting the system) or were general to the acquisition (e.g. the system shall be in balance).

Careful examination of the 1,100 evaluation factors indicates is it possible to assign a large number of the factors to five categories. The two categories with the most factors are assigned the labels **performance** and **usability** based on the purpose of the factors grouped together, with more than 20% and 30%, respectively, of the factors associated with them. About 14% of all the requirements are associated with aspects of **reliability**, availability and

serviceability. 8% of the factors were associated with functions and tests that assess the **effectiveness** of systems being able to provide resources to the workload. Finally, the smallest grouping was **consistency**, the capability of the system to produce consistent results. The working definitions of these five categories for the remainder of this work are:

- **Performance** – factors that contribute to how fast or how much work can be done on the system. The types of factors in this category are performance rates and amounts and/or capacities of equipment.
    - Examples
        - *"final system must achieve, on average, one and one-half (1½) TFlops/s of measured, sustained system performance over the first three (3) year period"*
        - *"peak performance of at least sixty teraFLOPs/s and a peak plus sustained performance of at least eighty teraFLOPs/s on the two SSP marquee benchmarks"*
- **Effectiveness** – factors that relate to managing workflow on the systems so the users of the system are able to get high performance results.
    - Examples
        - *"Demonstrate that XX batch jobs can be simultaneously active on at 95% of the compute*

nodes", where XX is a specific set of batch jobs for that site.

- *"Prioritized IO operations"*
- *"priority job scheduling"*
- *" priority group scheduling"*

- **Reliability** – factors that relate to functions, features or services that make the systems reliable and serviceable are in this category.
  - Examples
    - *"final system must achieve, on average, one and one-half (1½) TFlops/s of measured, sustained system performance over the first three (3) year period"*
    - *"peak performance of at least sixty teraFLOPs/s and a peak plus sustained performance of at least eighty teraFLOPs/s on the two SSP marquee benchmarks"*
- **Consistency** – factors that relate to providing consistent results, both in terms of reproducibility of answers and time to do a given amount of work.
  - Examples
    - *"a 30 availability test to complete within a 90 day window… controlled by purchaser"*
    - *"Be able to prepare new releases for installation without interrupting normal service"*

- **Usability** – this category is for features that make the system usable to both the end customers (mostly scientists and engineers) and system managers.
    - Examples
        - *“Operating system based on UNIX*
        - *“Applications Programming Interface (API) to the hardware registers or counters which can be used to measure performance attributes of applications”*
        - “*Provide a debugger [for] Serial, Parallel, Profiler [for applications]”*

These five categories represent more than 84% of all the evaluation factors across all the RFPs, and an even larger percentage of the technical factors. They also include virtually all the tests – either benchmarks or other types of tests – which the RFPs specify.

Two other categories are used to capture the factors that are not specific or not technical.

- General – factors that apply across categories
- Other – factors that are not in the other categories. These may relate to facilities requirements, personnel issues and business factors.

Finally, there is one more category of evaluation factors – **Cost** – that all the RFP's handle differently. Cost is a separate factor in itself – but is not specified in most of the RFPs. Instead, proposers provide the costs of the systems and associated services and cost is used in conjunction with the other factors to evaluate the systems. Cost varies in importance in evaluations – with one site reporting the cost as only 10% of the criteria they used to decide on the system, and technical factors was 90% (Simard 2003). In other cases, cost can be up to 50% of the determination.

For most factors, there is a strong linkage to a primary category but for some factors, it is possible to group them into more than one category. Take, for instance, several RFPs that had the phrase "*All processors shall be identical"* or similar wording. For this analysis, this factor was placed in the consistency category, but it could also be placed in performance or usability, since identical CPUs improve performance of most parallel codes and, at the same time, make programming in parallel more straightforward. Having a consistency category solves the problem of primacy for such factors.

2.4.1 Minimum/Mandatory/Baseline Requirements

In the tables below, the number of requirements for each category is expressed as M.S where M is the number of major requirements. S is the number of significant, but Sub-Major level requirements

Area	RFP-1	RFP 2	RFP 3	RFP 4	RFP 5	RFP-6	RFP-7	RFP-8	RFP-9	RFP-10	RFP-11	RFP-12
Performance	4	25	21	5	4	8	5	10.3	1	8	27.5	27.24
Effectiveness	2	4.1	3	2	1	1	1	2	0	3	7	12.3
Reliability	2	6	8	2	2	0	0	5	0	10	13	26.3
Consistency	3	0	0	2	0	1	0	2	0	0	0	1
Usability	3	23	24	3	3	1	2	10.6	0	6.5	19	48.34
General	2	8	6	1	1	1	1	1	0	5	5	3
Other	1	13	30	3	4	0	1	0	0	7	12	2
Total	17	79.1	92	18	15	12	10	30.9	1	39.5	83.5	119.85
Award and delivery date(s)	2006/2007	2006 /2006	2008 /TBD	2006 /2010 expected	2002 /2002	1999 /2000	2005 /2005	2004 /2004	2004	2002 /2005	2004	2002 /2002

Table 2-2: Mandatory Major and Sub-Major Evaluation Factors

2.4.2 Desired/Performance/Non-Mandatory

RFP / Area	RFP-1	RFP 2	RFP 3	RFP 4	RFP 5	RFP-6	RFP-7	RFP-8	RFP-9	RFP-10	RFP-11	RFP-12
Performance	8	3	0	1	3	4.2	3	0	28	2.1	36	0
Effectiveness	3	0	0	1	1	5	1	0	8.2	0	26	4.3
Reliability	7	0	0	2	5.4	4	4	0	15.7	1	29	2
Consistency	3	0	0	0	0	1	1	0	1	0	4	0
Usability	10	1	1	7	5	7	5	0	58.1	1	56	11.7
General	3	1	0	2	2	1	1	0	4	2	8	3
Other	1	3	1	4	8	3	3	0	1	1	33	0
Total	35	8	2	17	24.4	25.2	18	0	116	7.1	192	21.7

Table 2-3: Desired Major and Sub-Major Evaluation Factors

The categories that have the largest number of factors are Performance and Usability, while Consistency has the smallest number of factors.

Category	Total Number of factors	Percent of All Factors
Performance	268	21.8%
Effectiveness	123	10.0%
Reliability	184	14.9%
Consistency	19	1.5%
Usability	357	29.0%
General	61	5.0%
Other	131	10.6%

Table 2-4: Breakdown of Factors by Category

Several of the RFPs analyzed were from NERSC, so the question arises as to whether NERSC RFPs unduly influence the categorization. Another concern might be that three of the RFPs have more than 100 factors, with two RFPs over 275 factors. Do these “large” RFPs overly influence the categorization?

Table 2-5: below shows that removing the NERSC RFPs produce very similar groupings as including them, so while they are consistent within the community, they are not unduly biasing the results. Likewise, looking at grouping factors for the 8 RFPs with 100 or less factors again shows similar results to the overall. In this case, the percent of factors associated with consistency increase, which may be due to fact the three largest RFPs are also some of the oldest, and the concern about consistency has only become evident in the recent time frame as systems of substantial scale being more susceptible to inconsistency the NERSC-4 RFP was the first RFP to have explicit factors for consistency (other organizations have now followed). Finally, reviewing of the three RFPs with more than 100 factors shows general agreement with the categorization of the other RFP groupings.

Category	Number of Factors of RFPs that are not related to NERSC	Percent of All Factors of RFPs that are not related to NERSC	Number of Factors of RFPs that have less than 100 factors	Percent of All Factors of RFPs that have less than 100 factors	Number of Factors of RFPs that have more than 100 factors	Percent of All Factors of RFPs that have more than 100 factors
Performance	221.00	23.4%	120.00	25.5%	148.00	22.0%
Effectiveness	105.00	11.1%	31.00	6.6%	92.00	13.7%
Reliability	151.00	16.1%	62.00	13.2%	122.00	18.1%
Consistency	8.00	0.8%	13.00	2.8%	6.00	0.9%
Usability	331.00	32.9%	123.00	26.2%	234.00	34.8%
General	46.00	4.9%	38.00	8.1%	23.00	3.4%
Other	103.00	10.9%	83.00	17.7%	48.00	7.1%

Table 2-5: This table shows consistency in factor categories independent of site or number of factors.

Another question in this regard is how consistent is the categorization of factors across RFPs. Table 2-6: shows the percentage of evaluation factors in each category relative to the total number of factors for that RFP. Again, there is striking similarity between the categories, with almost all RFPs having about twice the number of factors in the performance and usability areas as in the other areas.

RFP / Area	RFP 1	RFP 2	RFP 3	RFP 4	RFP 5	RFP-6	RFP-7	RFP-8	RFP-9	RFP-10	RFP-11	RFP-12
Performance	23%	32%	22%	17%	16%	36%	29%	33%	23%	21%	24%	14%
Effectiveness	10%	6%	3%	9%	5%	15%	7%	5%	8%	6%	12%	14%
Reliability	17%	7%	9%	11%	26%	10%	14%	13%	17%	21%	15%	16%
Consistency	12%	0%	0%	6%	0%	5%	4%	5%	1%	0%	1%	0%
Usability	25%	27%	27%	29%	19%	21%	25%	41%	47%	23%	27%	28%
General	10%	10%	6%	9%	7%	5%	7%	3%	3%	13%	5%	2%
Other	4%	18%	33%	20%	28%	8%	14%	0%	1%	15%	16%	1%
	2006/ 2007	2006 /2006	2005 /2008	2006 /2011	2002 /2002	1999 /2000	2005 /2005	2004 /2004	2004	2002 /2005	2004	2002 /2002

Table 2-6: Percent of Evaluation Factors by category. Note the highest percentages are in Performance and Usability.

2.5 Cross Cut Groupings

While the major categories of factors are consistent across the analyzed RFPs, it is possible to make other groupings. As an example, three other categories that have increased in attention recently are; security, facilities and accounting. Assessing RFPs by these categories shown in Table 2-7.

RFP / Area	RFP 1	RFP 2	RFP 3	RFP 4	RFP 5	RFP-6	RFP-7	RFP-8	RFP-9	RFP-10	RFP-11	RFP-12
Security	1		1	1		1			2		11	
Facilities	1	2	2	4	2			1	2		12	
Accounting									1		2	3
Percent of Total												
Security	0.0%	0.0%	0.1%	2.6%	0.0%	2.6%	0.0%	0.0%	1.5%	0.0%	3.9%	0.0%
Facilities	2.0%	2.4%	0.3%	10.5%	9%	0.0%	0.0%	0.0%	1.5%	0.0%	4.3%	0.0%
Accounting	2.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.8%	0.0%	0.7%	1.1%

Table 2-7: Cross Cut of other groupings of factors

One thing to notice is there is less commonality in this grouping than in the PERCU categories. Another aspect is little commonality exists across systems of similar size, except for the smallest systems. Some of the sites that do sensitive work have few security requirements – leading one to presume they use other mechanisms for assure system integrity.

2.6 Chapter Conclusion

The evaluation factors used by different organizations to assess computer systems can be classified in a consistent manner, regardless of the size and scope of the system being evaluated. Technical classification into five categories of Performance, Effectiveness, Reliability, Consistency and Usability account for 84% of the evaluation factors used in the RFPs, and almost all the technical factors. Many factors besides performance used for purchasing decisions.

Hence, the PERCU classification is reasonable for evaluation factors sites to use to evaluate HPC systems for acquisition. Since acquisitions must evaluate the entire system rather than only sub-functions, PERCU is a useful way to organization system evaluation methods. The remainder of the work investigates effective ways to assess systems in each category.

Chapter 3: Sustained System Performance Method

3.1 Chapter Summary

The class of performance evaluation factors is clearly important as indicated in the analysis described in Chapter 2. This chapter explains the Sustained Systems Performance (SSP) method, which provides a process for evaluating system performance across any time frame. SSP is a simple but powerful method in the sense that it can be applied to any set of systems, any workload and/or benchmark suite, and for any time period. SSP measures time-to-solution across different application areas and it can be used to evaluate absolute performance and performance relative to cost (in dollars, energy or other value propositions).

While the formula development in this chapter is meant to be complete, it should not be intimidating since the SSP method can be described in a straightforward explanation in Section 3.2 below.

3.2 The Basic SSP Concept

"Work done per unit time" (Culler and Singh 1999)is a well accepted method to compare computer system performance. SSP uses one or more benchmarks to compare the performance of two or more systems using time-to-solution and a well defined classification of "work" as the primary performance indicators. Each benchmark has one or more problem sets which, together with the benchmark, determine a unique test. Each test has a

total operational count (Floating Point Operations – Flops from a single reference system - are used in this work, but other operations can be used if appropriate) that can be determined with code profiling or static analysis. For each test on each system, a per processor performance rate is determined by measuring and/or projecting the time-to-completion of the test on the system. Per processor rate of each test is determined by dividing the total operation count by the runtime of the test and then again by dividing the number of processors used in the test.

Once the effective per processor performance rate is determined for each test, a single per processor performance rate can be calculated with a composite function such as an arithmetic or geometric mean. To determine the Sustained System Performance of a system, the composite per processor rate is multiplied by the total number of computational processors in the entire system.

Systems may change in time due to upgrades and/or additions. Comparing two or more systems may also be complicated because technology may be introduced at different times. If one needs to compare systems that become available at different times, or will change over time, the SSP for each system can be determined for each time period or phase. For each phase of the system, the SSP multiplied by the length of the phase gives an indication of the amount of work the system can perform a work during that time period. This is called a system's *potency* or the system's

potential to perform work. The potency for each phase of a system can be added together, essentially integrating the capability of the system(s) to perform work over a defined time period, giving the potential of the system to handle the work (as represented by the tests) over its targeted time period. Once the potency of the system is determined, it can be compared either in absolute terms or relative to cost functions such as initial hardware costs, total cost of ownership (TCO) or energy usage. Using the potency of a system relative to its costs can determine the value of one system for comparison with other systems.

The end result is a method that assesses any system over any time frame. The workload representation can be arbitrarily complex or simple and span any number of application domains. The systems can be compared for performance and/or price performance using SSP.

Assessing performance of systems is a well studied field that stems from the earliest days of computers. The use of benchmarks to represent the work a system is expected to support is an accepted approach and the details of the many variants of benchmarking will not be repeated here. Instead the reader is pointed to many of the references listed.

One key feature, that is almost universally agreed upon is that the best way to assess how appropriate a system is at solving a problem is how long the system takes to solve a real problem. SSP is a meta-benchmark since it is a unique method that provides a composite view of a system's capability to

perform work. SSP is flexible because it can use any set of benchmarks or tests, of any scale or number, to evaluate performance, taking into account the fact systems and technology evolve over time. As shown in this chapter and the next, SSP can be implemented to measure and compare time-to-solution across different usage domains.

This chapter uses a running example to illustrate the definitions and formulas discussed below. The data for the example is similar to actual data provided in evaluating systems for purchase, but has been simplified. The next chapter has a more complete, almost real world, example of using the SSP method, with data refined from an actual procurement of systems, to illustrate the method in full.

For this simplified example, consider an evaluation of systems 1, 2, and 3, under consideration for purchase for a fixed amount of money. To understand the performance of these systems, the targeted workload is represented by three benchmarks; A, B, and C. The benchmarks may be industry standard, simple kernels, pseudo applications or full applications; it does not matter for the example. Each benchmark has a single problem data set associated with it that runs at a fixed concurrency (e.g. number of tasks), but the concurrencies do not have to be the same across applications. Assume this example uses three problem sets, one for each benchmark.

3.3 Buying Technology at the Best Moment

Whenever someone buys technology based electronic components, the decision making process is influenced by Moore's Law (Moore 1965). This is true whether the technology is consumer electronics, personal computers or a supercomputer. The fundamental question for the purchaser is:

"If I wait a little bit longer, I can get a system with better performance for the same cost. Should I wait, even if I am losing getting some work done in the meantime?"

This question becomes critical when selecting HPC systems due to high cost and long lead times of these very large systems. Determining the time to purchase a single system from a single vendor may be a simpler question because one only has to assess how long to wait and how much better the later system would be. However, just going to the local computer store shows the simple case never exists because different systems from different vendors with different options are available at different times. How does someone decide what to do?

The primary motivation of this chapter and the following one is to discuss how the Sustained System Performance Test (NERSC SSP Project 2004) addresses the "when to buy" question as well as "what to buy" question. SSP does this by providing a quantitative assessment of measured performance over time. If the test components are properly chosen, SSP provides a good expectation of the on-going – or sustained – performance the system produces. Furthermore, SSP can be used to represent a complex workload

with a metric that is meaningful to the users of the technology. The metric can be made arbitrarily complex or left simple, so it can represent a wide range of usage and circumstances.

While the SSP concept can be applied to almost any technology, we will focus here on how SSP can be used to evaluate HPC Systems. This chapter discusses the SSP approach and the methods used to calculate SSP. The next chapter will investigate the use of SSP in both theoretical analysis and in real world purchases.

3.4 Good Benchmark Tests Should Serve Four Purposes

Benchmark tests are approximations of the real work a computer system can accomplish. In other words, benchmark tests estimate the potential of computer systems to solve a set of problems.

Benchmark tests have four distinct purposes. Benchmark tests are made up of computer programs and the input data sets that state a problem for the program to solve. One set of computer code can exhibit different behavior based on the problem being solved and the parameters involved. Each purpose of the benchmark tests influences the selection and the characteristics of the benchmarks as well. The four purposes of benchmarks are:

1. Evaluation and/or selection of a system from among its competitors.

2. Validating the selected system actually works the way it is expected to operate once a system is built and/or arrives at a site. This purpose may be more important than the first and is particularly key when systems are specified and selected based on performance projections rather than actual runs on the actual hardware.
3. Assuring the system performance stays as expected throughout the systems lifetime (e.g. after upgrades, changes, and regular use.)
4. Helping guide future system designs.

The sophistication of the approximation represented by the benchmarks depends on the fidelity needed to represent the true workload. Later in this chapter, there is a more complete discussion of the relationship between benchmark selection and their ability to represent a workload. Comparison measures can range from assessing the peak capability of the hardware to using many full application tests with a range of problem data sets. In between full applications and simple peak rates are simple performance probes (e.g., LINPACK (Dongarra 1985) and GUPS (Earl, Willard and Goldfarb 2000)), micro kernels (ping-ping (MVAPICH Ping-Pong Benchmark n.d.) , stream (Streams Benchmark n.d.) , Livermore FORTRAN Kernels (McMahon 1986) , etc.) and limited and pseudo applications (e.g. NAS Parallel Benchmarks (Bailey, Barszcz, et al. March 1994) - also known as the NPBs, SPEC (SPEC Benchmarks 2000) , etc).

Most reports in the literature discuss only the first of these four purposes benchmarks play in the life of a system. The majority of tests claim to do well

on the first goal and possibly one other of the goals, but few are effective in all. Take as an example the widely discussed LINPACK benchmark that is used to determine in the HPC Top 500 List (Top 500 List 2008). LINPACK (LINPACK Download 2008) is a single test that solves Ax=b with dense linear equations using Gaussian elimination with partial pivoting. For matrix A, that is size M x M, LINPACK requires 2/3 M^2 + $2M^2$ operations. The latest LINPACK benchmark implementation, High Performance LINPACK (HPL), can run on any number of processors, but in order to provide enough work to each processor, the size of the A matrix has to increase, not only taking more memory, but increasing the wall clock time of the run more than linearly. This is *memory constrained scaling* "which is attractive to vendors because such speed ups are high" (Culler and Singh 1999). In order to keep scaling high as much work per processor as possible has to loaded into the system's memory. The amount of memory used grows at $O(N^2)$; the run time to do the work grows at $O(N^3)$. So for a system such as the NERSC Cray XT-4 with a ~39,000 cores and ~80 TB of aggregate memory, a single run of LINPACK may take 17-20 hours on the entire system. Often multiple runs need to be done to determine an accurate LINPACK value, but such a measure cannot be done often.

LINPACK is used to evaluate computer systems, as demonstrated by the Top 500 list, and is occasionally used as a specification, thereby serving the first purpose of a benchmark. In a limited way, LINPACK is used to validate whether a system meets expectations at the time of arrival. The limited use of

LINPACK for this purpose is due to the fact that LINPACK correlates very well with peak performance, but there are many applications whose performance does not correlate with LINPACK. Further, running LINPACK at scale takes a long time. LINPACK also has little to add to future architectural improvements, except possibly as a regression test to insure architectures continue to do well with dense, cache friendly computations. Since LINPACK only partially addresses purpose 1 and 2, and does not address 3 or 4, it is a less meaningful indicator of how well as system is able to process work.

3.5 Definitions for SSP

There are a few global definitions along with detailed examples of key terms and equations. The reader may initially skip this section and refer to it to understand equations later if they choose.

Definition	Explanation
Flops/s	This work follows the recommended notation found in *The Science of Computer Benchmarking* (Hockney 1996) by Roger Hockney which recommends the following conventions: • Ops is the plural of operation and • Ops/s as the rate of the number of operations performed per second. Because the examples and data of this work come from scientific computation, floating point operations are the norm, so • *Flops* indicates the amount of operations and • *Flops/s* indicates the rate of operations. Of course standard prefixes of M for Mega, G for Giga and T for Tera are used as is appropriate. The author acknowledges there are common uses of Ops as a rate rather than an amount in the literature.
CPU = core = processor	For the sake of simplicity, we will use the term *processor* or *CPU* as our concurrent element for now, where processor is identical to a single core for multi-core chips.

	Some processors are created with component “sub processors”. For example, take the case of the Cray X1/X1e. Most people use it as one high performance vector processor, called a *Multi-streaming Processor* (MSP) (Cray X1E n.d.) (Boucher, et al. 2004) However, the MSP is made up of four *Single Stream Vector Processors*, each identical, that can be used in unison or independently. Hence, for this system, a CPU can be defined as either a Multi-streaming Processor or a Single Stream Vector Processor, as long as the analysis is consistent. Another architectural variation is a standard processor integrated with accelerators. An example of this is the IBM/Sony/Toshiba “Cell” processor introduced in 2005 (Gschwind, et al. 2005) (Hofstee 2005). The cell processor consists of a Power PC integrated on chip with eight *Synergistic Processing Elements* (SPEs). Each SPE can execute an independent instruction stream. Further, a Cell processor may be combined with a commodity processor such in the LANL “Roadrunner” system (CNet 2006) which uses one AMD Opteron processor in conjunction with a Cell processor. In this case, the integration is not necessarily on-chip. Other systems proposed integrating commodity processors with graphics processing units and/or FPGAs. In the cell case, there are several choices regarding the definition of CPU. One is to define the CPU as the integrated unit Power PC and the SPEs (the “cell”). This would be a homogenous unit. Alternatively, one could define multiple CPU types – the PPC, the SPE, and the non-embedded commodity process, providing a heterogeneous set of CPUs. The SSP methodology allows either definition as one CPU or as a number of independent CPUs. If the latter, then the system will be treated as a heterogeneous system. Being able to use the appropriate definition for a processing element and to allow a system to have different types of processing elements is important in making the evaluation method apply to a large range of applications.
Heterogeneous System	A computing system with more than one type of processor architecture and/or processor speed combinations available for computational work.
Homogeneous System	A computing system with only one type of processor architecture/speed available for computational work.

Table 3-1: Sustained System Performance Definitions

3.6 Constants

The tables below have all the indices for each constant or variable. For the sake of simplicity, one or more indices may be omitted if it does not cause confusion for that part of the analysis.

Definition	Explanation
I	The number of different applications used in the evaluation.
J_i	The number of data sets each application executes. The number of problem data sets may be different for different applications. So J_i is the number of data sets for application i for $1 \le i \le I$. If all applications have the same number of data sets, then just J is used.
S	The number of systems in the evaluation.
K_s	The number of evaluation periods/phases for system s, where $1 \le s \le S$. K may be different for different systems. k_s is the k^{th} phase of system s. K_s is the total number of phases for system s. $1 \le k_s \le K_s$
$A_{s,k}$	The number of different processor types in system s during phase k. An example of a system with different processors is the IBM/Sony/Toshiba Cell processor which may be considered to have two processors types. Another example could be a system with a mix of AMD and Intel processors. Later it will be used to index processor types, so $1 \le \alpha \le \mathbf{A_{s,k}}$
$L_{s,k}$	The number of cost components for system s during each phase k. Cost components are used to develop a cost function that can later be used to determine the value of a system. For example, a system may have costs for hardware, software, maintenance, electrical power, etc. Not all costs apply to every phase, since some phases may involve only software improvements.

Table 3-2: SSP Definitions for SSP Constants

3.7 Variables

Definition	Explanation	Units Generic [Used in this work]
$f_{i,j}$	The total reference operation count of application i executing data set j. The reference operation count is determined once for each application/data set combination. It can be determined by static analysis or by using hardware/software performance counters on a reference system. Examples of tests that have reference operation counts pre-defined are the NAS Parallel Benchmarks, LINPACK and the Livermore FORTRAN Kernels. Using a single reference count for all systems tested results in an evaluation of time-to-solution being compared. It is recognized that executing application i with data set j on a given system may actually generate a higher or lower operation count on a given system. It may be appropriate that a specific application be used on one or more data sets in order to fully evaluate a system. The typical HPC measure is Floating Point operations (Flops). Other metrics may be the appropriate work output measure. (E.g. for climate	Operations [Flops, MIPs, Ops] [Simulated Years]

	it could be in simulated years). The total amount of work operations may change for different concurrencies and on different systems. $f_{i,j}$ is the reference amount of work done on a chosen reference system and thus remains constant for the analysis for every *i* and *j*.	
$\mathbf{m}_{\alpha,i,j}$	The concurrency of application *i* executing data set *j* for processor type *α*. Often, the concurrency of an application/data set is the same as that used to set the reference operation count. It is important to note the concurrency element is not fundamental to the evaluation, but, being able to consistently determine the total number of concurrent elements for a test suite is important for overall performance and value assessment. For the most part, concurrency is the number of CPUs an application *i* specified to use to solve a given data set *j*. The concurrency can be allowed to be different for a test if the processors are dramatically different. For example, a Cray system might have both scalar and vector processors with a factor of four differences in performance. It may be appropriate to adjust the concurrency due to the large difference in performance for the same data set. If the same data set is used for different concurrencies across all systems, it is treated as a different data set (a new *j,* so to speak) so there is a one-to-one mapping of operations count for problem *j* and concurrency for data set *j*. Likewise, if an application is used more than once with two different concurrencies, it can be considered another application. For some analyses, it is possible to allow different concurrencies on each system s for a given *i,j*. The advantage is providing the opportunity to run an application of optimal scalability. While the SSP method works with this approach since per processor performance can be calculated for each system, it typically adds complexity to use the same data to understand individual application performance. For systems where there is a choice of the defining CPU in different manners, such as with vector processors or cell processor technology, concurrency is defined relative to the choice of the different CPU components.	[Processors]

$\mathbf{a_{\alpha,i,j}}$	The work done in each concurrent unit for application *i* for data set *j* on processor type *α*. Equation 3-1: Work per processor $$a_{\alpha,i,j} = \frac{f_{i,j}}{m_{\alpha,i,j}}$$ Note that $a_{i,j}$ does not imply what $a'_{i,j}$ would be if the test were run with a different concurrency $m'_{i,j}$	Ops per Concurrent Schedulable Unit [Flops per Processor]
$\mathbf{t_{s,k,\alpha,i,j}}$	The time-to-completion for application *i* running data set *j* on processor type *α*. There is timing and hence performance for each phase *k* of each system *s* for each processor type. Most references recommend wall clock time as the best time with good reasons, but others (user CPU time, overall CPU time) are frequently seen.	Time [seconds]
$\mathbf{p_{s,k,\alpha,i,j}}$	The per processor performance of application *i* executing data set *j* on processor type *α* on system s during phase *k*. Equation 3-2: Per processor performance $$p_{s,k,\alpha,i,j} = \frac{a_{\alpha,i,j}}{t_{s,k,\alpha,i,j}} = \frac{f_{i,j}}{\left(m_{\alpha,i,j} \times t_{s,k,\alpha,i,j}\right)}$$ **Important Note** Since $f_{\alpha,i,j}$ is a fixed value based on a reference system, as long as the concurrency $m_{\alpha,i,j}$ is held constant for all systems, the performance per processor for system *s*, running application *i*, with test case *j*, relies solely on the time it takes to solve the problem on system *s*. Hence, this is a comparison of time-to-solution for the application.	Ops/(proc*sec) [Flops per second per processor]
$\mathbf{w_i}$	The weight assigned to application *i*. Weights may be the proportion of time the application used in the past or the amount of time or resources the corresponding science area is authorized to use. w_i values do not change for a given evaluation. If w_i is the same for all *i*, then the analysis is unweighted. Later in this work there is a significant discussion on whether and when weights should be used. The SSP methodology accommodates either weighted or unweighted approaches.	
$\mathbf{W}$	The one dimensional array of length *I* containing the weights w_i	
$\mathbf{P_{s,k,\alpha}}$ $\mathbf{P^*_{s,k,\alpha}}$	An array of all $p_{s,k,\alpha,i,j}$ for a given phase *k,* processor type α and system *s* $P^*_{s,k,\alpha}$ is a sub-set of $p_{s,k,\alpha,i,j}$ where $p_{s,k,\alpha,i,j}$ are selected by some criteria. For example, the criteria	

	may use only the results from the largest data set runs.	
$\tau_{s,k}$	The starting time of evaluation phase *k* for system *s*	Days, months, years, etc. [months]
τ_o	The start of the evaluation period. This can either be arbitrarily set or it can be set to the earliest period for any of the systems being evaluated.	Days, months, years [months]
τ_{eval}	The length of the evaluation period. τ_{eval} is set to be the same for all systems in a given evaluation.	Days, months, years [months]
τ_{max}	The end time for the evaluation period. $\tau_{max} = \tau_o + \tau_{eval}$	Days, months, years [months]
$N_{s,k,\alpha}$	The number of computational processors of type α in system *s* during evaluation period *k*. $N_{s,k,\alpha} \geq m_{\alpha,i,j}$, In other words, there have to be enough processors of type α in the system to run application *i* executing data set *j* for processor type α. A system may consist of different types of processors indicated by α. Systems that run parallel applications, frequently have $\alpha = 1$ at least for the processors expected to do the computational workload. In this case, the α notation may be omitted. Such a system is called homogeneous. Alternatively, if $\alpha > 1$, the system is called heterogeneous.	
$c_{s,k,\ell}$	The cost of factor ℓ for time phase *k* of system *s*, where $1 \leq \ell \leq L_{s,k}$. A cost factor may be the cost of hardware, the cost of software, the on-going costs of operating a system, etc. It is possible to evaluate cost in great detail and therefore have large numbers of cost factors, such as the cost and number of each memory DIMM and each CPU chips making $L_{s,k}$ very large. However, system costs are often presented as aggregated prices for a system. Indeed, most vendors make it difficult to know the true cost factors of a system. Again, SSP can accommodate any degree of detail.	Currency [Dollars]
$\Phi(W, P_{s,k,\alpha})$	The composite per processor performance function for the set of weights *W*, and the set of per processor performance *P* for performance phase *k* for processor type α on system *s*. This will be discussed in detail later.	Ops/(proc*sec) [Flops/s per processor]
$SSP_{s,k}$	Sustained System Performance for system *s* for phase *k*. Equation 3-3: Sustained System Performance for system	Operations /time [Flops/s]

	s during phase k $SSP_{s,k} = \sum_{\alpha=1}^{A_{s,k}} \left(\Phi \left(W, P_{s,k,\alpha} \right) * N_{s,k,\alpha} \right)$	
Potency$_s$	The potential of a system *s* to produce useful work results over a period of time. Potency was chosen for this term since it means "capacity to be, become, or develop (Dictionary.com n.d.);" Equation 3-4: A system's potency is a reflection of its ability to do productive work. $Potency_s = \sum_{k=1}^{K_s} SSP_{s,k} * [\min(\tau_{s,k+1}, \tau_{\max}) - \min(\tau_{s,k}, \tau_{\max})] \forall \tau_{s,k} \leq \tau_{\max}$ There will be more discussion of systems with phases in Chapter 4.	Operations [Flops] Note: it is possible to consider Potency as a rate [Flops/s] multiplied by a time period [day, months, year, etc.]. Hence, Potency can also be expressed more as integrated performance over time. [e.g. TFlops/s * Years or GFlops/s * Months]
Cost$_s$	The cost of system *s*. Cost is composed of different components $c_{s,k,l}$. Equation 3-5: Costs are used for setting value of a solution $Cost_s = \sum_{k=1}^{K_s} \sum_{l=1}^{L_{s,k}} c_{s,k,l}$	Currency [Dollars]
Value$_s$	The ratio of potency to cost for system *s* Equation 3-6: Value of a solution is its potency relative to its cost $Value_s = \frac{Potency_s}{Cost_s}$ Potency and Cost are influenced by the number of processors, $N_{s,k,\alpha}$, but the degree of influence cannot be assumed to be equivalent for many reasons including economies of scale and business strategies.	Operations Cost of Resources [Flops/$]

Table 3-3: Variables and Formulas for determining the SSP.

3.8 Running Example Part 1 – Applications

Our running example has 3 benchmarks; A, B, and C, each with one problem set. Hence I = 3. Each benchmark uses only the Message Passing Library (MPI) (MPI Forum 1993) calls so there is a mapping of one MPI task to one CPU. Since each benchmark has only one data set, J = 1, it is omitted for clarity. Three systems are evaluated, each with uniform processors, so S = 3. α= 1 and is omitted for clarity.

Table 3-4 below summarizes the benchmarks' characteristics. The operation counts can be determined in a variety of ways, but most systems today have a utility to count the number of operations for a problem run.

Application	Total Operation Count, *f* GFlops	Concurrency, *m* Processors	Amount of work done in each task, *a*. GFlops/processor
A	549,291	384	1430
B	310,842	256	1214
C	3,143,019	2,048	1535

Table 3-4: This table shows the basic performance characteristics for the three benchmarks in our example

Before examining the proposals for the systems, it is possible to assume these benchmarks were run on a baseline system, such as NERSC's Power 3 system Seaborg. Table 3-5 shows the per processor performance of these runs.

Application	Wall Clock Runtime, *t* Seconds	Per Processor Performance, *p* GFlops/s/processor
A	42,167	0.034
B	9,572	0.127
C	12,697	0.121

Table 3-5: Baseline performance of benchmarks on an existing system.

3.9 Aligning the Timing of the Phases

Evaluations should have a consistent time period for all systems. Since systems could likely arrive at different timings, aligning these periods is necessary for a fair comparison.

Additionally, for each system, there can be more than one phase of system evolution. A phase is characterized by the system having different components and/or additional components that make the potency different than the previous phase. The cost of each phase, as well as the potency maybe different as well. For the evaluation there is a period set, τ_{eval} to limit the length of the evaluation period. τ_{eval} is often related to how long the technology is to be used. NERSC uses 36 months and DOD-HPC Modernization program uses four years (High Performance Technology Insertion 2006 (TI-06) 2005).

Systems are unlikely to be delivered at exactly the same time and it is not equitable to use the individual system delivery times as the starting point since the price/performance ratio of a system delivered later is almost certainly less than one delivered earlier – all other things being equal.

However, another consequence of later delivery is lost computing opportunities. A simple thought experiment shows why this is important. Suppose an organization has two choices: have a 100 teraflop/s system deployed in January 1, 2007 or a 500 teraflop/s system deployed in January 2, 2012. Assume they have the same real cost and assume sustained performance of the 2012 system is also five times that of the 2007 system. A valid question could be along the lines of "How long is it before the organization has the same potency as it will in April 1, 2013?" The answer is it takes 1.25 years of use of the 2012 system to provide the same potency as the 2007 system. The result of an organization choosing to wait for the system with the better price/performance ratio is no computing for 5 years at all. It then takes 1.25 years to make up the lost computing power.

So, are the two systems equal in value to the organization? It depends on the organizational goals and many factors such as whether the same number of people can gain the same insight in 25% of the wall clock time and whether there are opportunities lost by having no computing for 5 years. It is clear the phase boundaries have to be adjusted for each system in the evaluation in order to fairly represent the value of different solutions. The adjustments that have to be made are straightforward. Algorithms for the adjustments are shown in Table 3-3: Variables and Formulas for determining the SSP.

First, the system with the earliest delivery date is identified, which sets the starting point, τ_o to be beginning of the evaluation period. It may be that the

organization needs a system by a certain time, so the evaluation period has to start no later than a given deadline. In this case, τ_o can set to the earliest of the arrival date of the first system or to a specific no-later-than deployment time set by the evaluators – whichever is earliest.

Not only can different systems arrive for use at different times, it may be that the best overall solution is to have systems evolve through time, based on optimizing different price points. A way of describing the timing of each phase a system goes through, which is $\tau_{s,k}$ is needed. For each system *s*, there will be one or more stages, indicated by *k*.

Because solutions cannot wait indefinitely for a system to provide solutions, the evaluators set the ending time τ_{max} to end the evaluation period. τ_{max} is specified as $\tau_{max} = \tau_o + \tau_{eval}$. Once τ_o and τ_{max} are determined, all the systems solutions being evaluated need to be adjusted to that interval. This is done by adjusting the starting period of all the systems to τ_o, and assigning the performance and cost for a system during this period to be 0. Likewise, for the systems whose delivery would take them past τ_{max}, no performance or cost is counted for that system.

FiguresFigure 3-1andFigure 3-2 show the impact of these adjustments. Figure 3-1 shows two systems being evaluated. System 1 arrives and is deployable before System 2 and has a single phase. System 2 arrives and is deployed after System 1 and has an improvement part way through the evaluation process, triggering the second phase.

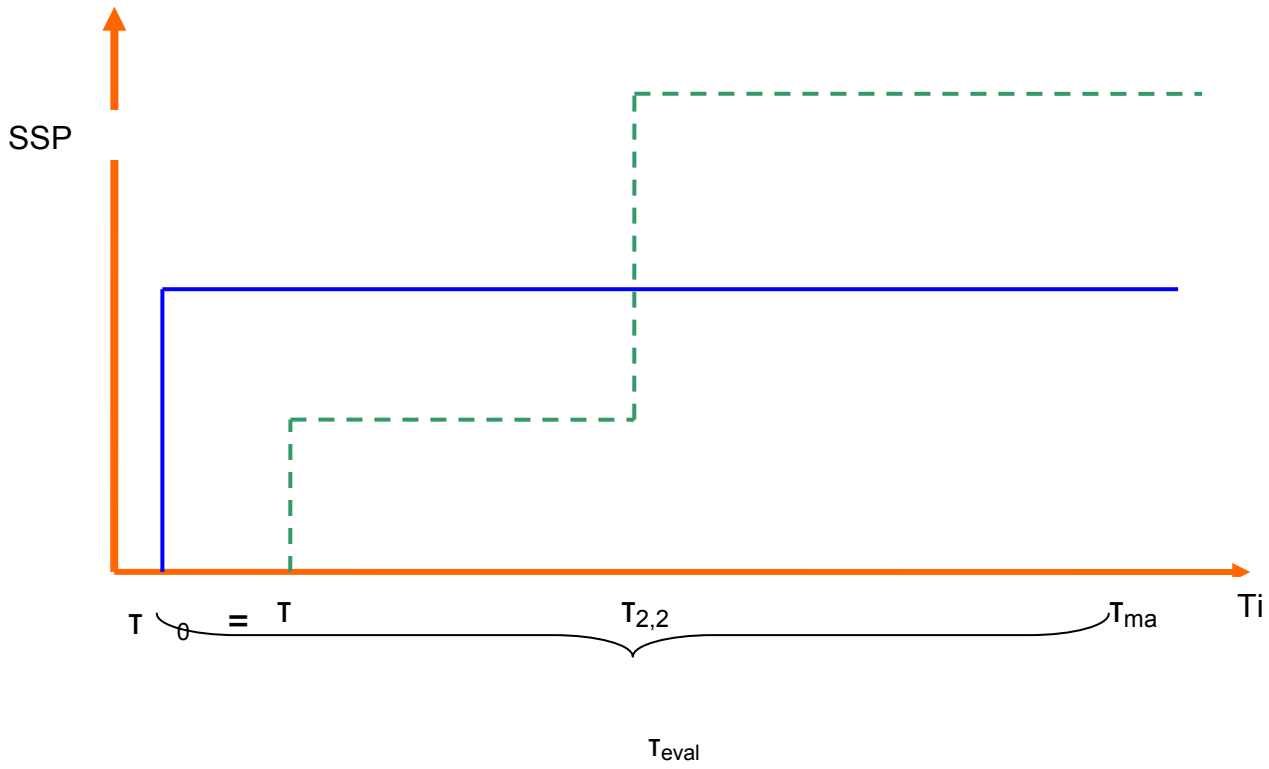


Figure 3-1: The proposed deployment time and SSP of two systems.

Assuming System 1 is deployed before any time limitation such as T_{NLT}, the deployment of System 1 defines T_o for both systems. Since System 2 arrives after T_o, the performance and cost for System 2 is set to 0 until it arrives. The end of the evaluation period is also set based on System 1's deployment time. After these adjustments are used the evaluation periods are shown in Figure 3-2.

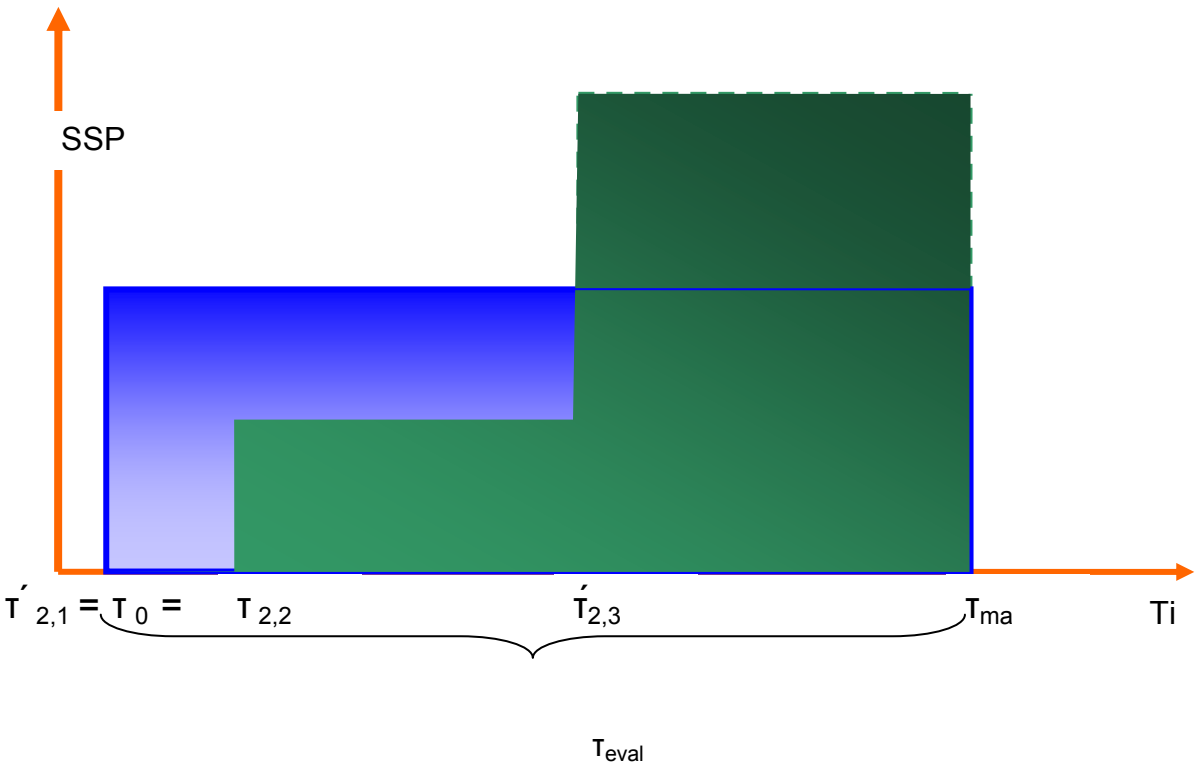


Figure 3-2: SSP performance chart after periods are aligned. For clarity τ́2,k replaces τ2,k

3.10 Running Example Part 2 – Systems

Our running example assumes three different systems are being evaluated. System 1 has a high per processor performance, but each processor is relatively expensive. Because it is under development, it can only be delivered 9 months after System 2. System 2 is a system that is currently available for immediate delivery and consists of processors that are modest in performance, but are also relatively inexpensive. System 3 is a system that has per processor performance that is close to System 2. While part of the system can be delivered immediately, ¾ of the system is delayed

by 4 months due to manufacturing backlog. Furthermore, the first ¼ of System 3 will perform 20% slower until the rest of system is delivered. System 3's per processor cost is lower than either System 1 or System 2.

For simplicity, assume other aspects of the systems are identical except for the cost. Note the "cost" can be any calculation – from only initial hardware cost to complete total cost of ownership. The costs in the example approximate 6 years TCO for this scale system.

Table 3-6 indicates the system parameters for the running example. The time period of the evaluation is set at 36 months – a typical time period during which large systems have the most impact.

System	Delivery Delay (Months)	Number of Compute Processors	Cost (Dollars)
System 1	9	9,000	$59,000,000
System 2	0	10,000	$68,000,000
System 3			$57,000,000
- Phase 1	0	3,500	
- Phase 2	6	14,000	

Table 3-6: Specifications of solutions being considered

From this information one cannot determine the solution that is the best value. The benchmark runtimes for the three systems are shown in Table 3-7, and the resulting per processor performance in Table 3-8.

Runtimes in seconds of Benchmarks on each System	A	B	C
System 1	3810	1795	2303
System 2	3598	2010	3170
System 3			
- Phase 1	4930	2622	2869
- Phase 2	4103	2185	2391

Table 3-7: Benchmark Runtimes in Seconds for Three Systems

Per Processor Performance in GFlops/s of Benchmarks on each System	A	B	C
System 1	.375	.676	.666
System 2	.398	.604	.484
System 3			
- Phase 1	.290	.463	.535
- Phase 2	.348	.556	.642

Table 3-8: Per processor performance of three benchmarks

3.11 The Composite Performance Function $\Phi(W, P)$

The composite performance function can be chosen in different ways. There are many possible functions, depending on the data and goals. Several may be appropriate for a given evaluation. Which functions are appropriate for different performance measures is an on-going discussion and is covered in (Bucher and Martin 1982), (Flemming and Wallace 1986), (Smith 1988), (Lilja 2000), (Patterson and Hennessey 1996), (John and Kurian, Performance Evaluation and Benchmarking 2006), (Helin and Kaski 1989), and (Mashey 2004). The SSP method can use any appropriate composite. Hence, this section does not try to do an exhaustive study of all possible functions, but rather is a general discussion of several likely functions and how to implement them.

Recall w_i and $P_{s,k,\alpha}$ as defined above, and a composite function are used to calculate an overall performance rate. Some typical composite functions are Arithmetic Mean, Harmonic Mean and Geometric Mean – all of which can use either weighted or unweighted data. More advanced statistical functions could be used such as the *t* test or an Analysis of Variance (Ostle 1972).

Equation 3-7, Equation 3-8, and Equation 3-9 show the implementation of the three more common means. If $w_i = 1$ for all i, then the means are unweighted.

$$\Phi_{wAM} = \frac{\sum_{i=1}^{I} w_i \sum_{j=1}^{J_i} p_{i,j}}{\sum_{i=1}^{I} w_i}$$

Equation 3-7: Weighted Arithmetic Mean

$$\Phi_{wHM} = \frac{\sum_{i=1}^{I} \sum_{j=1}^{J_i} w_i}{\sum_{i=1}^{I} \sum_{j=1}^{J_i} \frac{w_i}{p_{i,j}}}$$

Equation 3-8: Weighted Harmonic Mean

$$\Phi_{wGM} = \left(\prod_{i=1}^{I} \prod_{j=1}^{J_i} \left(p_{i,j}^{w_i} \right) \right)^{\left(\frac{1}{\sum_{i=1}^{I} \sum_{j=1}^{J_i} w_i} \right)}$$

Equation 3-9: Weighted Geometric Mean

3.12 SSP and Time-to-Solution

The number of operations a computer uses to solve a given problem varies dramatically based on the computer architecture, its implementation, and the algorithm used to solve the problem. While this has been the case from the dawn of computing, the problem of deciphering how much work is done by different systems has gotten harder with time. Early performance results on distributed memory computers were so notorious for providing misleading information that it prompted Dr. David Bailey to publish his *Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers* paper (D. Bailey 1991) in 1994. In this paper, 7 of the 12 ways (ways 2, 3, 4, 6, 8, 9, and 10) relate to using measurements that are misleading for comparisons that vary depending on the computer system being used or doing a subset of the problem. New processors and increasing scale compound this problem by causing more and more basic computer operations to be done for a given amount of usable work. Different algorithms, programming languages and compilers all influence performance in addition to the computer architecture and implementation (Patterson and Hennessy 2007).

Many performance evaluators recognize that time-to-solution is the best – maybe only – meaningful measure of the potential a computer provides to address a problem. If the system is to be used for a single application, assessing time-to-solution is relatively straight forward. One takes an

application, tests the systems with one or more problem sets, and compares the time it takes for the application to solve the problem. An example of this approach is a metric commonly used by the climate modeling community which is the number of simulated years per unit of wall clock time. Weather forecasting has similar metrics – the wall clock time it takes to produce a given forecast. Chemical and materials simulations could have a similar measure – the time it takes to simulate a compound with a fixed number of atoms, for example. In these cases, as long as the algorithm and problem (the same resolution, the same precision of modeling physical processes, etc.) remains unchanged, there is a fair comparison based on time-to-solution.

A more challenging, but more commonly occurring situation is when computer systems are evaluated for use with different applications and/or domains because there is no common unit of work that can be used for comparison. That is, it is not meaningful, for example, to compare the number of years a climate simulation produces in a given wall clock time to the size of a protein that is folded in a chemical application. Similarly, if the problems or physics the algorithms represent change within an application area, the comparison of the amount work done is not clear cut. Finally, if the implementation of an application has to fundamentally change for a computer architecture, the number of operations may dramatically different.

It is common, therefore, for performance evaluators to use the number of instructions executed per second (also known as operations per second) as measured on each system, or other less meaningful measures methods (e.g. peak performance, Top-500 lists, etc.), This approach leads to easily misunderstanding comparative results.

SSP addresses the issue of in precise or multiple factor work measures since the operation count used in the calculation of SSP is fixed once for the benchmark test and is based on the problem solution executing on a reference (presumably efficient) system, albeit a test case that is reasonable for the system sizes under evaluation. If the problem concurrency is also fixed, the only invariant is the time the test takes to run to solution on each system.

To show SSP is a measure of time-to-solution if the operation count is based on a reference count, consider the following. For each combination of an application and problem set, i,j, the number of operations $f_{i,j}$ is fixed as is the concurrency, $m_{i,j}$.

$$p_{s,i,j} = \frac{f_{i,j}}{m_{i,j} * t_{s,i,j}} = \frac{f_{i,j}}{m_{i,j}} * \frac{1}{t_{s,i,j}}$$

Equation 3-10: The per processor performance for a test depends on the time to complete that test

For simplicity, but without loss of generality, assume an unweighted composite function. Again for simplicity, use the standard mean and assume

all the processors in a system are homogeneous and there is a single phase. The per processor performance can be expressed as:

$$P_s = \frac{(\sum_{i=1}^{I}\sum_{j=1}^{J}\frac{f_{i,j}}{m_{i,j}} * \frac{1}{t_{i,j}})}{I} = \frac{(\sum_{i=1}^{I}\sum_{j=1}^{J}\frac{f_{i,j}}{m_{i,j}})}{I} * \frac{(\sum_{i=1}^{I}\sum_{j=1}^{J}\frac{1}{t_{i,j}})}{I}$$

Equation 3-11: Per processor performance for a system depends on time-to-solution

The equation of SSP performance between two systems, *s* and *s'* with the same number of computer processors, N, can be expressed as follows:

$$\frac{SSP_s}{SSP_{s'}} = \frac{N * \sum \frac{f_{i,j}}{m_{i,j}} * \frac{1}{t_{s,i,j}} \Big/ (I*J)}{N * \sum \frac{f_{i,j}}{m_{i,j}} * \frac{1}{t_{s',i,j}} \Big/ (I*J)} = \frac{\sum \frac{f_{i,j}}{m_{i,j}} \Big/ (I*J)}{\sum \frac{f_{i,j}}{m_{i,j}} \Big/ (I*J)} * \frac{(\sum \frac{1}{t_{s,i,j}}) \Big/ (I*J)}{(\sum \frac{1}{t_{s',i,j}}) \Big/ (I*J)} = \sum \frac{t_{s',i,j}}{t_{s,i,j}}$$

Equation 3-12: Comparing SSP values is equivalent to comparing time-to-solution

Hence, SSP compares the sum of the time-to-solution for the tests. From this, it is clear that if the number of processors is different for the two systems, then the SSP is a function of the time-to-solution and the number of processors. If the systems have multiple phases, the SSP comparison is dependent on the time-to-solution for the tests, the number of processors for each phase and the start time and duration for each phase. This can be further extended for heterogeneous processors and/or for different composite functions without perturbing the important underlying concept the SSP compares time-to-solution across different systems.

3.13 Attributes of Good Metrics

There are benefits of using different composite methods, based on the data. The approach of using the harmonic mean was outlined in a 1982 Los Alamos technical report (Bucher and Martin, 1982). It should be noted that at the time, the workload under study was a classified workload with a limited number of applications. In fact, the authors dealt with “five large codes”. The paper outlines the following process.

1. Workload Characterization: Conduct a workload characterization study using accounting records to understand the type and distribution of jobs run on your systems.
2. Representative Subset: Select a subset of programs in the total workload that represent classes of applications fairly and understand their characteristics. This included the CPU time used, the amount of vectorization, the rate of Floating Point Operation execution and I/O requirements.
3. Weighing the influence of the Applications: Assign weights according to usage of CPU time of those programs on the system.
4. Application Kernels: Designate portions (kernels) of the selected programs to represent them. These kernels should represent key critical areas of the applications that dominate the runtime of the applications.

5. Collect Timing: Time the kernels on the system under test using wall clock time.
6. Compute a Metric: Compute the weighted harmonic mean of kernel execution rates to normalize the theoretical performance of the system to a performance that would likely be delivered in practice on the computing center's mix of applications.

Bucher and Martin were focused on the evaluation of single processors – which was the norm at the time. As stated, the implementation of this methodology suffers from some pragmatic problems.

1. It is difficult to collect an accurate workload characterization given that many tools for collecting such information can affect code performance and even the names of the codes can provide little insight into their function or construction (the most popular code, for instance, is 'a.out').
2. Most HPC Centers support a broad spectrum of users and applications. The resulting workload is too diverse to be represented by a small subset of simplified kernels. For example, at NERSC, there are on the order of 500 different applications used by the 300-350 projects every year.
3. The complexity of many supercomputing codes has increased dramatically over the years. The result is that extracting a kernel is an enormous software engineering effort and maybe enormously difficult. Furthermore, most HPC codes are made up of combinations of fundamental algorithms rather than a single algorithm.

4. The weighted harmonic mean of execution presumes the applications are either serial (as was the case when the report was first written) or they are run in parallel at same level of concurrency. However, applications are typically executed at different scales on the system and the scale is primarily governed by the science requirements of the code and the problem data set.
5. This metric does not take into account other issues that play an equally important role in decisions such as the effectiveness of the system resource management, consistency of service, or the reliability/fault-tolerance of the system. The metric also is not accurate in judging heterogeneous processing power within the same system – something that may be very important in the future.

John and Eeckhout indicate the overall computational rate of a system can be represented as the arithmetic mean of the computational rates of individual benchmarks if the benchmarks do not have an equal number of operations. Nevertheless, there are attributes of making a good choice of a composite function. Combining the criteria from (Smith,1988) and (Lilja, 2000) provides the following list of good attributes.

- Proportionality – a linear relationship between the metric used to estimate performance and the actual performance. In other words, if the metric increases by 20%, then the real performance of the system should be expected to increase by a similar proportion.

    - A scalar performance measure for a set of benchmarks expressed in units of time should be directly proportional to the total time consumed by the benchmarks.
    - A scalar performance measure for a set of benchmarks expressed as a rate should be inversely proportional to the total time consumed by the benchmarks.
- Reliability means if the metric shows System A is faster than System B, it would be expected that System A outperforms System B in a real workload represented by the metric.
- Consistency so that the definition of the metric is the same across all systems and configurations.
- Independence so the metric is not influenced by outside factors such as a vendor putting in special instructions that just impact the metric and not the workload.
- Ease of use so the metric can be used by more people.
- Repeatability meaning that running the test for the metric multiple times should produce close to the same result.

SSP reflects these attributes. There has been a series of papers debating which mean is most appropriate for performance evaluations for at least 16 years. In fact there have been disagreements in the literature about the use of the geometric mean as an appropriate measure. Note that the SSP method allows any mean, or other composite function, to be used equally well in the calculation and different means are appropriate for different studies. Hence,

this section discusses the attributes of different means and the summary of the papers, but does not draw a single recommendation. That depends on the data and the goal of the study.

The arithmetic mean is best used when performance is expressed in units of time such as seconds and is not recommended when performance is expressed as performance ratios, speedups (Smith, 1988) or normalized values (Flemming and Wallace, 1986). The arithmetic mean alone may be inaccurate if the performance data has one or more values that are far from the mean (outlier). In that case, the arithmetic mean together with the standard deviation or a confidence interval is necessary to accurately represent the best metric. (John 2004) concludes the weighted arithmetic mean is appropriate for comparing execution time expressed as speedups for individual benchmarks, with the weights being the execution times.

The harmonic mean is less susceptible to large outliers and is appropriate when the performance data is represented as rates. The unweighted harmonic mean for a system phase can be expressed as total operation count for all benchmarks divided by the total time of all benchmarks as shown in Equation 3-8.

Use of geometric means as a metric is not quite as settled. (Smith, 1988) says it should never be used a metric, while (Flemming and Wallace, 1986) indicates it is the appropriate mean for normalized numbers regardless of how they were normalized. They also note it addresses the issue of data that has

a small number of large outliers. This paper also points out the geometric means can be used for numbers that are not normalized initially, but when the resulting means are then normalized to draw further insight.

(Mashey, 2004) examines the dispute and identifies that there are reasons to use all three means in different circumstances. Much of the previous work assumed some type of random sampling from the workload in selecting the benchmarks. This paper notes that geometric means have been used in many common benchmark suites such as the Livermore FORTRAN Kernels and the Digital Review CPU 2 (Digitial Review 1998) benchmarks. This paper organizes benchmark studies into three categories, and each has its appropriate methods and metrics. The first and most formal category is WCA (Workload Characterization Analysis), which is a statistical study of all the applications in a workload, including their frequency of invocation and their performance. WCA is equivalent to the methodology outlined in Bucher and Martin. This type of analysis provides a statistically valid random sampling of the workload. Of course, WCA takes a lot of effort and is rarely done for complex workloads. WCA also cannot be done with standard benchmark suites such as NPB or SPEC. While such suites may be related to a particular workload, by their definition, they cannot be random samples of a workload.

The Workload Analysis with Weights (WAW) is possible after extensive WCA because it requires knowledge of the workload population. It can predict workload behavior under varying circumstances.

The other type of analysis in (Mashey, 2004) is the SERPOP (Sample Estimation of Relative Performance of Programs) method. In this category, a sample of a workload is selected to represent a workload. However, the sample is not random and cannot be considered a statistical sample. SERPOP methods occur frequently in performance analysis. Many common benchmark suites including SPEC and NPB, as well as many acquisition test suites, fit this classification. In SERPOP analysis, the workload should be related to SERPOP tests, but SERPOP does not indicate all the frequency of usage or other characteristics of the workload.

The impacts of the assumptions in early papers (fixed time periods, random samples, etc.) that discuss the types of means are not valid for SERPOP analysis. Because of this, the geometric mean has several attributes that are appropriate for SERPOP analysis. In particular, (Mashey, 2004) concludes geometric means are appropriate to use for ratios since taking ratios converts unknown runtime distributions to log-normal distributions. Furthermore, geometric means are the appropriate mean for SERPOP analysis without ratios when there are many reasons the distribution of a workload population is better modeled by a log-normal distribution.

3.14 Running Example Part 3 – Holistic Analysis

For our running example, the arithmetic mean will be used to calculate the SSP and Solution Potential.

System Evaluation using SSP	Average Per Processor Performance GFlops/s	System SSP using the mean of the three benchmarks GFlops/s * Months	Solution Potential GFlops/s * Months	Solution Value GFlops/s-Months/ Million $
System 1	.573	5,155	139,180	2,359
System 2	.495	4,953	178,294	2,622
System 3 - Phase 1 - Phase 2	 .429 .515	 1,503 7,214	225,426 9,017 216,426	3,955

Table 3-9: Per processor performance of three benchmarks

While System 1 has the highest per processor performance, because it is delivered quite a bit later than the other systems, it has the lowest potential and value. System 2, even though it is delivered at full scale earliest, has the middle value. System 3, with two phases clearly has the highest value for the evaluation using these benchmarks.

3.15 Chapter Conclusion

The SSP method is flexible and comprehensive, yet is an easy concept to understand. It uses the most meaningful measure for computer users to make its comparisons – time-to-solution. The method works with any set of performance tests and for both homogeneous and heterogeneous systems. The benchmarks can have as many data sets as needed and be scaled to the desired degree of precision and effort.

Chapter 4: Practical Use of SSP for HPC Systems

This chapter examines the use of the SSP method to assess systems in practice. It also examines the improvements made to SSP based on applying it to evaluations.

4.1 Chapter Summary

This chapter provides a number of examples using the SSP method to evaluate and assess large systems. It traces the evolution of the SSP method over a 10 year effort as it became more sophisticated and effective. Large HPC systems are complex and evaluated/purchased only once every three to five years. Hence 10 years gave the opportunity to have and assess four generations of SSP. As part of the observations of SSP, it can be seen that the SSP method gives both the purchaser and the supplier of systems protection. The supplier has freedom to adjust the schedule of deliverables and the purchaser is protected by a guarantee of a fixed amount of performance delivered in a certain time period. The degree of adjustments can be constrained as well, so it is possible to arrange incentives for early delivery or delivery of more effective systems.

Chapter 3 discusses the SSP method for overall performance assessment that is one method to evaluate the Performance of a system. While SSP is not the only measure used to assess a system's potential to solve a set of problems, it is one of the few that, if properly constructed, can be used for all

four purposes of benchmarks. Section 3.8 discussed a simplified example of a problem. This chapter takes the SSP approach from the previous chapter and examines the use of SSP in different real world evaluations and selection issues in a variety of circumstances.

4.2 A Real World Problem, Once Removed

It is not possible to disclose the details of actual procurement submissions or evaluations since the information is provided by vendors to the requesting organization for evaluation and selection and is considered proprietary. However, it is possible to craft a summary that is based on real world information from such a procurement that is sufficient to properly illustrate the use of SSP.

Imagine an organization evaluating large scale systems for a science or engineering workload. The organization determines functional and performance requirements and creates a benchmark suite of applications and other tests. It then solicits and receives information from potential vendors as to how well their technology meets the requirements and performs according to the benchmarks. In many cases the response is a combination of actual measurements and projections. For simplicity, assume the only distinguishing characteristics under evaluation are the specified benchmark performance on a per processor basis shown in Table 4-1. They are the set of $p_{,s,k,\alpha,i,j}$ that was defined in Table 3-2: SSP Definitions for SSP Constants

There are five proposed systems (S= 5). Five application benchmarks are used to calculate the SSP, so I = 5. While the applications run at medium to high concurrency, they operate at different concurrencies. Each application has been profiled on a reference system so its operation count is known for particular problem sets. In this case, each application has one problem, so J_i = 1 for this discussion. Further, assume these systems are composed of homogeneous processors so α= 1, so it, too, is set to 1.

As defined in Table 3-3, in order to calculate the per processor rate of the applications, $p_{s,k,1,i,1}$, the total operation count of the benchmark is divided by the concurrency of the benchmark to give the average per processor operation count and then divided again by the wall-clock runtime of the benchmark. Four of the five systems proposed had phased technology introduction, with each of these having K_s=2.

The cost data includes basic hardware and software system costs and the operating and maintenance costs for three years from the delivery of the earliest system. In order to protect the proprietary information provided by vendors, the cost data is expressed relative to the lowest cost proposed. Delivery times all are relative to the earliest system delivery and set to the number of months after the system with the earliest delivery time.

		System 1	System 2	System 3	System 4	System 5
Phase 1						
Application Benchmark 1	GFlops/s per Processor	0.31	0.20	0.74	N/A	0.22
Application Benchmark 2	GFlops/s per Processor	0.40	0.30	1.31	N/A	0.06
Application Benchmark 3	GFlops/s per Processor	1.35	0.17	0.64	N/A	1.19
Application Benchmark 4	GFlops/s per Processor	1.00	2.34	5.99	N/A	1.12
Application Benchmark 5	GFlops/s per Processor	0.49	0.51	1.02	N/A	0.45
Delivery	Months after earliest delivery	3	0	6	N/A	0
Number of Processors		768	512	512	N/A	512
Phase 2						
Application Benchmark 1	GFlops/s per Processor	0.31	0.19	0.74	0.10	0.22
Application Benchmark 2	GFlops/s per Processor	0.40	0.34	1.31	0.30	0.06
Application Benchmark 3	GFlops/s per Processor	1.35	0.16	0.64	0.39	1.19
Application Benchmark 4	GFlops/s per Processor	1.00	1.54	5.99	0.92	1.12
Application Benchmark 5	GFlops/s per Processor	0.49	0.26	1.02	0.14	0.45
Delivery	Months after earliest delivery	12	22	18	3	6
Number of Processors		1536	1024	1408	5120	2048
Cost Factor	Relative cost among proposals	1.65	1.27	1.27	1.16	1.00

Table 4-1: Per processor performance, p, for each system, phase and benchmark for a hypothetical system purchase. These responses are anonymized and adjusted from actual vendor responses for major procurements. Systems 1, 2, 3, and 5 are proposed to be delivered in two phases. System 4 is a single delivery. The per processor performance of five application benchmarks is shown. The systems would be delivered at different times. The table shows the delivery date relative to the earliest system.

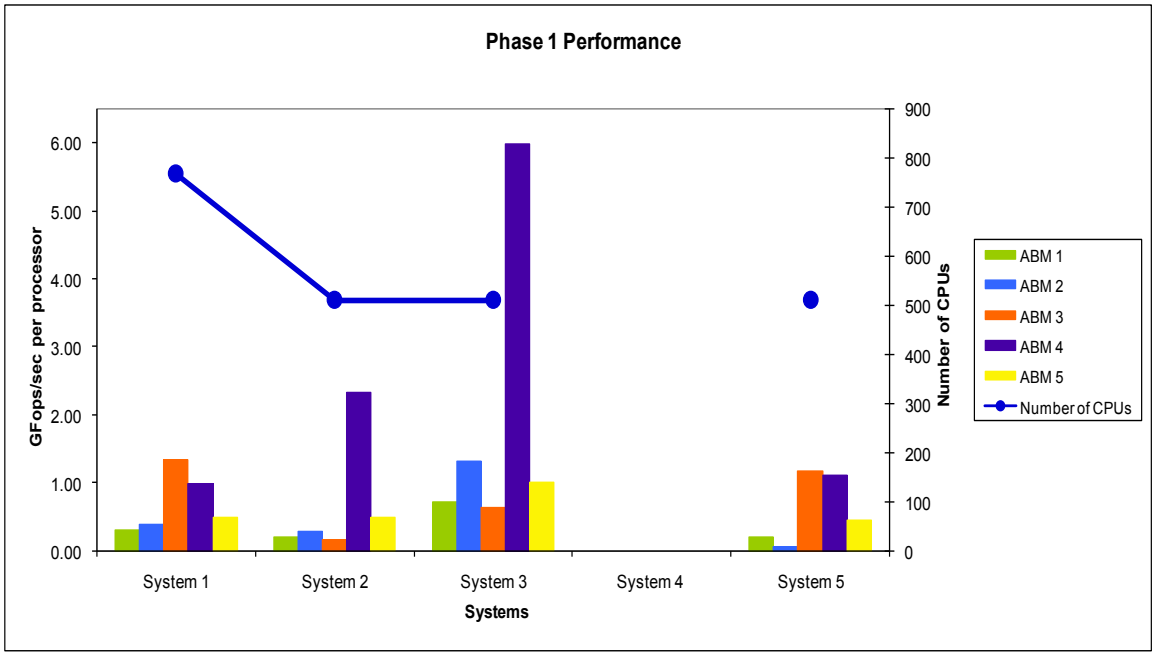


Figure 4-1: System parameters for Phase 1. Note System 4 is a single phase and is shown in the Phase 2 chart.

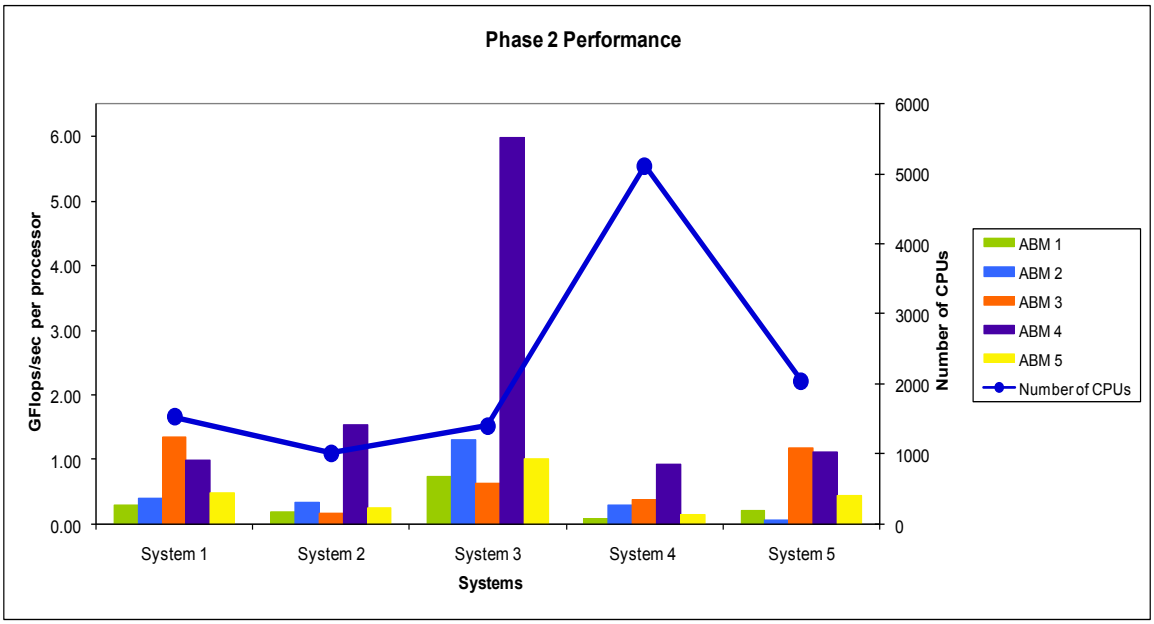


Figure 4-2: System parameters for Phase 2.

Figure 4-1 and Figure 4-2 show the same data as in Table 4-1, but in graphical form. The challenge of an organization is to use this data to decide

which system is the best value for the organization's mission and workload. As can be seen in Figure 4-1and Figure 4-2, the answer of which option provides the system with the best value is not obvious from the benchmark performance alone.

4.3 Different Composite Functions

As discussed in Chapter 3, different composite functions can be used for SSP calculations – including all three means. The best composite function to use depends on the data and the evaluation goals. Table 4-2 shows using geometric and arithmetic means and the impact they have on SSP Potency and value. Notice that the ordering of the system value is the same regardless of whether the arithmetic or geometric mean is used. For the rest of this example case, the unweighted geometric mean will be used for the composite function since the applications were selected are a non-random sample, so this is a SERPOP analysis as discussed in Section 3.13 above. This is equivalent to adding the integrated the per processor performance curves for each application times the number of processors for that phase, and then dividing by the number of applications.

		System 1	System 2	System 3	System 4	System 5
$Potency_s$ - Geometric Mean	GFlops GFlops/s*Months	26,486	9,474	41,074	45,959	24,587
$Value_s$ - Geometric Mean	Normalized (GFlops/s*Months)/$	564	271	1450	1201	781
$Potency_s$ - Arithmetic Mean	GFlops GFlops/s*Months	31,037	15,081	61,077	62,570	39,236
$Value_s$ - Arithmetic Mean	Normalized (GFlops/s*Months)/$	661	403	2,156	1,635	1,246
Ratio of Arithmetic vs. Geometric		1.17	1.49	1.49	1.36	1.60

Table 4-2: SSP Performance results using geometric and arithmetic means, and the impact on SSP Potency and Value.

4.4 Impact of Different Means

The relationship of means is HM ≤ GM ≤ AM (Selby 1968). Comparing the results of arithmetic mean and geometric mean show there are differences in the expected performance of the system. The ratio of performance between systems is not equal between means but in every case, the geometric mean is lower than the arithmetic for the system listed in Table 4-2. Furthermore, using the arithmetic mean, the order of best to worst price performance is Systems 3, 4, 5, 1 and 2. Using the geometric mean, the order is 3, 4, 5, 1 and 2. So the ordering of the system is preserved regardless of the mean used in this situation. In another example shown in Table 4-3, running the SSP-4 test suite (discussed in detail later in this chapter) on different technology systems at Lawrence Berkeley National Laboratory (LBNL) (systems named *Seaborg*, *Bassi* and *Jacquard*) and Lawrence Livermore National Laboratory (LLNL) (system name *Thunder*) using the arithmetic,

harmonic and geometric means changes the absolute value of the SSP, but does not change the order of performance estimates.

	Seaborg (LBNL)	Bassi (LBNL)	Jacquard (LBNL)	Thunder Cluster (LLNL)
Computational Processors	6224	888	4096	640
Arithmetic SSP-4	1,445	1,374	689	2,270
Geometric SSP-4	902	835	471	1,637
Harmonic SSP-4	579	570	318	1,183

Table 4-3: Another example of using different means that do not change the ordering of system performance

Since the ordering of the means is consistent, and the harmonic mean is less appropriate as a composite function for benchmarks that change their concurrency, the arithmetic or geometric means are used at NERSC and their affects are discussed in Sections 3.11 and 4.3. For the examples in Section 3.2, the arithmetic mean is used.

4.5 System Potency

Delivery of each system, as in our example, would occur at different times. Graphing the monthly values of the SSP for each system over time as shown in Figure 4-3 differentiates some of the systems. For example, System 2, despite having good per processor performance, had a low Potency since it has relatively few CPUs. To be the best value it would have to be 5 or 6 times less expensive than other alternatives. At the end of the evaluation period, System 3 provided the best sustained performance, followed by system 4. Yet, System 3 was upgraded after System 2 and 5, and System 4 had only a

single phase so it is not clear from this graph which system has the most potency, let alone the best value.

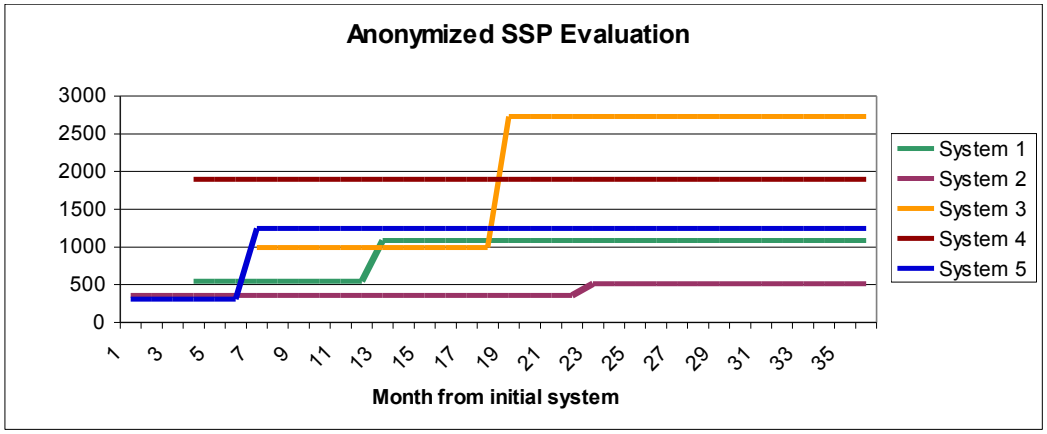


Figure 4-3: A graph of the example SSP value over time for the five systems. This is using the geometric mean as the composite function. The duration of the evaluation period is set by the evaluator. The starting date of the evaluation period can either be specified in an RFP or can be determined based on the first available system.

As a thought experiment, think about the situation where there are two system, and System 1 is twice as fast as System 2. In other words, it has an $SSP_{1,k} = 2 * SSP_{2,k}$. Assume further, System 1 is also twice as expensive System 2. The first thing to recall is having twice the $SSP_{s,k}$, does not mean the system has twice the Potency. In order to have twice the Potency, the two systems have to be available at the exact same time. This may be case with a commodity such as at a local computer store but is highly unlikely for HPC systems. Nonetheless, if the two systems arrived at identical times, and the Potency of System 1 was twice that of System 2, and the cost of System 1 was twice that System 2, they would have the same value with regard to the SSP. Further, in most major system evaluations, there are multiple cost

functions evaluated – for example the Initial System Cost and the total cost of ownership. Having all the cost functions identical is also unlikely.

But even if the Value of the two systems is exactly identical, that only reflects the performance component based on the selected benchmark tests. The overall value of the systems will almost certainly be different if the other aspects of PERCU are added to the evaluation. Or evaluators may choose to add second order benchmark tests, possibly to reflect use cases that are less important but to increase the accuracy of the SSP for the real workload as the "tie breaker".

4.6 Using Time-to-Solution in SSP

The way to calculate an SSP value using the SSP-5 tests is straightforward and illustrated below.

Application Tests Each application code has a data set the is specified for the concurrency) Tests	(a) Concurrency (MPI Tasks)	(b) Reference GFlop Count In this case, the reference system is the NERSC dual core Cray XT-4 system - *Franklin*	(c) Measured Wall Clock Time-to-solution (Sec) for the evaluated system	(d) Processing rate per core (GFlops/s)
CAM	240	57,669	408	0.589
GAMESS	1024	1,655,871	2811	0.575
GTC	2048	3,639,479	1492	1.190
IMPACT-T	1024	416,200	652	0.623
MAESTRO	2048	1,122,394	2570	0.213
MILC	8192	7,337,756	1269	0.706
Paratec	1024	1,206,376	540	2.182
Geometric Mean (GFlops/s)				0.7
Number of Compute Cores				*N*
System SSP				.7**N*

Table 4-4: Example calculation of a system's SSP value

Table 4-4 shows results for the SSP-5 suite with a typical runtime for the tests. As discussed above, the GFlop reference count (column b) is created on a reference system and does not change. In this case the reference system was the NERSC Cray XT-4 with dual core processors. For each test, the rate per core (column d) is the GFlop count divided by the number of tasks (column a) and divided by the time-to-solution (column c). Test rates per core are then composited, in this example with the geometric mean, determining the overall per core processing rate. The reason the per processor rate has to be determined for each code is the targeted concurrency for the reference problem is different for each test.

The per core processing rate is then multiplied by the number of compute cores in the system. In this case, if the system were to have 100,000 cores, the SSP value would be 70 TFlops/s. The SSP suites can have more or less tests and can be scaled to any degree.

Each test can be run multiple times for provide increased accuracy and/or to measure consistency of results, but that is an implementation decision left to the reader and is discussed more fully in Chapter 7:.

4.7 The Evolution of the NERSC SSP - 1998-2006

The SSP concept evolved at NERSC through four generations of system evaluations dating back to 1998. It serves as a composite performance measurement of codes from scientific applications in the NERSC workload including fusion energy, materials sciences, life sciences, fluid dynamics, cosmology, climate modeling, and quantum chromodynamics. For NERSC, the SSP method encompasses a range of algorithms and computational techniques and manages to quantify system performance in a way that is relevant to scientific computing problems using the systems that are selected.

The effectiveness of a metric for predicting delivered performance is founded on its accurate mapping to the target workload. A static benchmark suite will eventually fail to provide an accurate means for assessing systems. Several examples, including LINPACK, show that over time, fixed benchmarks become less of a discriminating factor in predicting application workload performance. This is because once a simple benchmark gains

traction in the community, system designers customize their designs to do well on that benchmark. The Livermore Loops, SPEC, LINPACK, NAS Parallel Benchmarks (NPB), etc. all have this issue. It is clear LINPACK now tracks peak performance in the large majority of cases. Simon and Strohmaier (Simon and Strohmaier 1995) showed, through statistical correlation analysis that within two Moore's Law generations of technology and despite the expansion of problem sizes, only three of the eight NPBs remained statistically significant distinguishers of system performance. This was due to system designers making systems that responded more favorably to the widely used benchmark tests with hardware and software improvements.

Thus, long-lived benchmarks should not be a goal – except possibly as regression tests to make sure improvements they generate stay in the design scope. There must be a constant introduction/validation of the "primary" tests that will drive the features for the future, and a constant "retirement" of the benchmarks that are no longer strong discriminators. On the other hand, there needs to be consistency of methodology and overlapping of benchmark generations so there can be comparison across generations of systems. Consequently, the SSP metric continues to evolve to stay current with current workloads and future trends by changing both the application mix and the problem sets. It is possible to compare the different measures as well so long running trends can be tracked.

SSP is designed to evolve as both the systems and the application workload does. Appendix C shows the codes that make up the SSP versions over time. Appendix B shows the SSP version performance results on NERSC production systems. Next is a description and evaluation of the four generations of the SSP metric.

4.7.1 SSP-1 (1998) - The First SSP Suite

The first deployment of the SSP method, designated SSP-1, was used to evaluate and determine the potency for the system called NERSC-3. This system was evaluated and selected in a fully open competition. SSP-1 used the unweighted arithmetic mean performance of the six floating point NAS parallel benchmarks (Bailey and el.al. 1991), in particular, the NAS Version 2.3 Class C benchmarks running at 256 MPI tasks. Additionally, the NERSC-3 acquisition used 7 full application benchmarks to evaluate systems but these applications were not part of the SSP-1 calculation. The NPB Benchmarks were used for the first SSP because they were well known to the vendor community, so the addition of the SSP method was less threatening to vendors, thereby encouraging participation. Further, analysis showed the NPBs had a relationship to a range of the applications of the time period. The applications benchmarks discussed in this section are shown in Appendix C.

4.7.1.1 Description of SSP-1

The NPBs were selected for SSP-1 for several reasons.

1. The procurement benchmark suite was developed using a 696 processor Cray T3E-900. Some applications selected from the workload were too large to get accurate reference instruction counts using the tools existing at the time. So an accurate reference flop count could not be established for several application/data set combinations that were used for the application benchmarks. Thus $f_{i,j}$ could not be accurately established for some values of i and j. On the other hand, the flop count for each NPB was analytically defined and validated by running on single processor systems for each problem size so $f_{i,j}$ was well known.
2. The NERSC-3 application benchmarks were a fixed problem size and vendors were allowed to choose the best concurrency for problem scaling in an attempt to determine the strong scaling characteristics of the proposed systems. The added complexity of each system using different concurrency was judged too risky for the first implementation of the SSP method. The fixed concurrency of the NPB codes was easier to implement for vendors for the new SSP method.
3. The NPB suite represented many fundamental algorithms used in the NERSC workload. For example, the NPB CG (Conjugate Gradient) test was similar to the 2D sparse matrix calculations in SuperLU (Li, Li and Demmel 2003) library test code and Quantum ChromoDynamics (QCD) application such as MILC (The MIMD Lattice Computation (MILC) Collaboration n.d.). The Fourier Transform (FT) test related to Paratec

(Canning, et al. 2000)((PARAllel Total Energy Code n.d.), which uses a global Fast Fourier Transform (FFT).

4. NERSC staff was familiar with the NPBs and could accurately interpret their implications for NERSC applications. Likewise, the NPBs were well understood by the vendor community and had proven easily portable to the potential systems, making their use less effort for vendors.

4.7.1.2 Assessment of SSP-1

The use of the NPBs as the SSP-1 metric was successful in several ways.

1. Vendors provided benchmark data for almost all the configurations proposed in part because the NPBs were well understood, easily portable and tested. The benchmarks had been ported to almost all the architectures and vendors were familiar with the implementation of the codes.
2. Feedback from the vendor community indicated[*] preference towards composite metrics such as SSP rather than a series of individual tests each with a performance requirement. This is because vendors were concerned about the number of individual benchmarks that represented many individual metrics – each with a risk of failure. Further, vendors indicated a willingness to agree to more aggressive composite goals since they had less risk than agreeing to perform for multiple discrete tests.

[*] Private communication from vendors during RFP debriefings.

3. As shown below, SSP-1 addressed a number of issues in making the system fully productive throughout its life time.

4.7.2 SSP-2 (2002) - The First Application Based SSP Suite

For SSP-2, performance profiling tools had advanced sufficiently to obtain accurate floating point and other operation counts of the application/problem set combinations at the scale needed. The SSP-2 metric used internal timing values of five application benchmarks: GTC (Lin, et al. 2002), MADCAP (Borrill 1999), MILC, Paratec, and SEAM (Taylor, Loft and Tribbia 2000). SSP-2 was based on a fixed reference operation count of all floating point operations in 5 benchmarks. All systems had homogeneous processors, so α = 1. In calculating SSP-2, one problem set was used for each application. All applications used the same concurrency and had to be run on the different systems at the specified concurrency.

4.7.2.1 Description of SSP-2

The composite function for SSP-2 was the unweighted harmonic mean, expressed as the total operation count of all benchmark/problem set combinations divided by the total time of all the application runs. All the operation counts $f_{i,j}$ for each application were summed. The per processor operation counts, $a_{\alpha,i,j}$, of each application i summed to 1,014 GFlops per processor.

4.7.2.2 Assessment of SSP-2

The use of application codes was successful and resulted in the user community having more confidence that the SSP-2 metric represented the true potential of the system to perform their applications. As indicated in Section 4.6, the evaluation of the system for which SSP-1 was developed also required a separate set of application test codes be run. SSP-2, because it used full application tests, meant vendors did not have to run a set of special codes for SSP-2 and a different set of codes for application testing. The fixed concurrency of the five codes made the SSP calculation simpler, but also led to some vendors failing to provide all the required data in their proposal because of issues of getting large benchmarking resources.

The biggest issue identified in the second generation of SSP was using the harmonic mean as the composite function. The harmonic mean resulted in essentially a weighted average, with the weight being the relative computational intensity of the applications. Computational intensity is the ratio of memory operations to arithmetic operations, with higher numbers indicating a code does more arithmetic operations per memory reference (Oliker, Bisaw, et al. 2006). Paratec was the most computationally intensive code in the SSP-2 test with a computational intensity almost three times that of the next code. Using an unweighted harmonic mean meant Paratec had more influence in the final SSP value than the other benchmarks, even though the materials science area represented about $1/10^{th}$ of the overall NERSC workload.

Fortunately, this imbalance did not have significant detrimental impact on user satisfaction with the selected system since Paratec was both computationally and communication intensive as it did significant communication with global Message Passing Interface (MPI) library calls and a global FFT as well as significant dense linear algebra. However, further analysis showed that if a different application code had been chosen that was not both computational and communication intensive, the potential existed to have a significant bias in the SSP-2 metric that was not intended. Hence, the SSP-3 version moved to the geometric mean to reduce this potential issue.

4.7.3 SSP-3 (2003) – Balancing Complexity and Cost

SSP-3 was intentionally scaled down* in order to select a modest size system. Selecting a benchmark suite has to take into account both the size of the target system and the expected amount of resources system vendors will be willing to use to provide results. The evaluators must balance these issues because the resources vendors invest to do benchmarking depends, in large part, on the eventual purchase price of the system. Since the targets system for the RFP was a system about 1/5th the dollar value of NERSC-3, the benchmark suite had to be correspondingly simpler and smaller – both in the number of codes and the concurrency of the codes.

* Because this and other systems were modest in science, NERSC refers to them as New Computing System (NCS) and a letter to designate their sequencing. So this RFP is referred to as NCSb.

4.7.3.1 Description of SSP-3

SSP-3 consisted of three applications and three NPB codes. The applications were CAM3 (Collins, et al. 2006), GTC and Paratec with a problem size that ran efficiently on 64 processors. SSP-3 had three NAS Parallel benchmarks: FT, Multi-Grid (MG) and Block Tri-diagonal (BT) from the NPB version 2.4 release using the Class D problem size running with a concurrency of 64 tasks.

4.7.3.2 Assessment of SSP-3

The SSP-3 codes were used to validate the delivered system and to assess sustained performance and consistency in performance. There was a very good response to the RFP in the number of proposals submitted. Also, all vendors provided all benchmark results. In some ways, the simplification in concurrency made SSP-3 too easy and too low in terms of concurrency to stress test the entire system when it was delivered. This meant other tests had to be used to detect deficiencies, which actually did exist and were rectified. Hence, simplifying the SSP codes to have vendors expend less up front effort made diagnosis of the system problems less efficient causing longer time between system delivery and full operation. This also added back end risk to NERSC of having less confidence the problems were identified before production. One example of these other tests that needed to be added is discussed in detail in Chapter 7 on consistency.

4.7.4 SSP-4 (2006) - SSP at Larger Scale

4.7.4.1 Description of SSP-4

SSP-4 consisted of the geometric mean of seven full application benchmarks: Madbench (Oliker, Borrill, et al. 2005), Paratec, CAM 3.0, GAMESS (Schmidt, et al. 1993), SuperLU, PMEMD (Amber n.d.) each with one large problem data set as the test problem. For SSP-4, the each benchmark ran at differing concurrency, ranging from 240 tasks to 2,048 tasks. SSP-4 was used for the NERSC-5 procurement. The goal of the average SSP performance for the first 36 months was 7.5 to 10 TFlops/s. SSP-4 was the first SSP to allow heterogeneous processors within a system to be considered.

4.7.4.2 Assessment of SSP-4

The SSP-4 used more application codes than any previous SSP, including one with a concurrency of 2,048. This seemed to have struck a good balance between the number and size of the benchmarks because all vendors provided complete data for the SSP applications – while several did not provide data for non-SSP applications.

4.7.4.3 SSP-4 Results

SSP-4 was used in the selection and acceptance testing of NERSC-5, which turned out to be a Cray XT-4 system. The first observation is that all bidders provided data for all SSP-4 applications, not just at the required

concurrency for the SSP-4 calculation, but at the other concurrencies as well. This may indicate that the mix of codes and concurrencies were a reasonable compromise between the needs of the facility and that of the vendors who offered systems.

SSP-4 was also the first time a Department of Energy's Office of Science site and the Department of Defense sponsored HPC Modernization Program coordinated the use of the same application benchmark, GAMESS with same problem sets. This cooperation was intended to reduce the effort for bidders to provide data and to be responsive the High End Computing Revitalization Task Force (HECRTF) Workshop report, which urged government agencies to coordinate benchmark requirements.

SSP-4 not only evaluated the systems offered and was used to validate the XT-4 during acceptance testing, but it also was used to evaluate two different Light Weight Operating System (LWOS) implementations*, at scale,

* On the XT-4 hardware, Cray offered two Light Weight Operating Systems (LWOS) – the Catamount Virtual Node (CVN) and the Cray Linux Environment (CLE) for the compute nodes. CVN is an extension of the Catamount kernel developed at Sandia National Laboratory for the XT family, originally created for the single core per node XT-3. It uses a master-slave implementation for the dual core XT-4. CVN provides minimal functionality, being able to load an application into memory, start execution, and manage communication over the Seastar Interconnect. Among many things, CVN does not support demand paging or user memory sharing, but does use the memory protection aspects of virtual memory for security and robustness, the latter to a limited extent. CVN does not support multiple processes per core and only has one file system interface.

The CLE (also commonly known as the Compute Node Linux kernel, which was Cray's pre-announcement designation) system, based on SUSE 9.2 during this comparison, separates, as much as practicable, computation from service. The dominant components of CLE are the compute nodes that run application processes. Service nodes provide all system services and are scaled to the level required to support computational activities with I/O or other services. The High Speed Network (HSN) provides communication for user processes and user related I/O and services.

Each CLE compute node is booted with a version of Linux and a small RAM root file system that contains the minimum set of commands, libraries and utilities to support the compute node's operating environment. A compute node's version of Linux has almost all of the services and demons found on a standard server disabled or removed in order to reduce the interference with the application. The actual demons running vary from system to system but include init, file system client(s), and/or

on the same hardware. This was the first such study at extreme scale of 19,320 cores.

Initially the NERSC XT-4 was delivered with the Cray Virtual Node (CVN) (Cray, Inc. 2007) light weight operating system (also known as a microkernel) operating system and SSP and other evaluation tests (ESP, full configuration tests, micro kernels, consistency, etc.) were used to assess it. After several months, the first release of the Cray Linux Environment (CLE) (Cray, Inc. 2007) light weight operation system emerged from the development process. The NERSC XT-4 was the first platform to move fully to CLE and remains the largest platform running CLE today.

The evaluation period for CVN and CLE each lasted six to eight weeks between the late spring and early fall of 2007. During this time, the LWOSs were progressively presented with more challenging tests and tasks, in all the areas of PERCU. The evaluation period can be considered as evolving through three phases that each has a different focus – albeit still approaching the system holistically. The first was a test of all functionality. Did the systems have all the features that were required and did they produce the expected (correct) results? The second phase was performance assessment when the

---

application support servers. CLE had specific goals to control OS jitter while maintaining application performance. CLE uses a user space implementation of the Sandia National Laboratory developed Portals interconnect driver that is multithreaded and optimized for Linux memory management. CLE also addressed I/O reliability and metadata performance.

systems were tested to determine how fast and how consistently they processed work. The third phase was an availability and performance assessment of the system's ability to run a progressively more complex workload while at the same time determining the general ability to meet the on-going system metrics. By the end of the third phase, a large part of the entire NERSC workload ran on the system, although with some limitations and a different distribution of jobs than is seen in production.

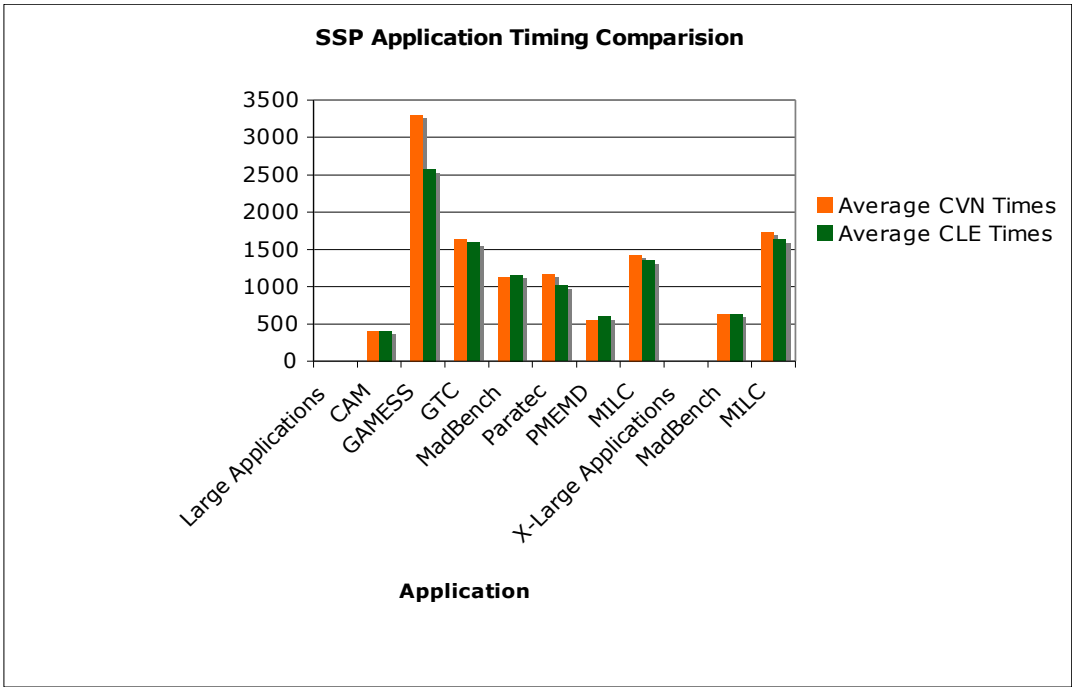


Figure 4-4: The SSP-4 application runtimes for two Light Weight Operating Systems running on the same XT-4 hardware. Note that most of the runtimes for CNL are lower than for CVN.

Figure 4-4 shows the SSP-4 application runtimes for both CVN and CLE running on the system hardware. The seven contributing applications to the SSP-4 metric are five large applications (CAM, GAMESS, GTC, Paratec and PMEMD) and two X-large applications (MadBench and MILC). The runtimes

for five of the seven SSP-4 applications are lower on CLE than on CVN. GAMESS shows the most improvement, 22%, followed by Paratec at almost 14%. The GAMESS' CLE runtime resulted from combining MPI and shared memory (SHMEM) communications in different sections of the code since MPI-alone or SHMEM-alone implementations ran longer on CLE than on CVN. Because GAMESS already supported MPI and SHMEM methods, it was not tremendously hard, albeit somewhat tricky to combine the two. The need to mix communication libraries resulted from different implementations of the Portals low-level communication library on CLE and CVN that changed the relative performance advantages between using the MPI and SHMEM Application Programming Interfaces (APIs). The improved Paratec timing was due in part to optimizing message aggregation in one part of the code. PMEMD showed better runtime on CVN by 10%.

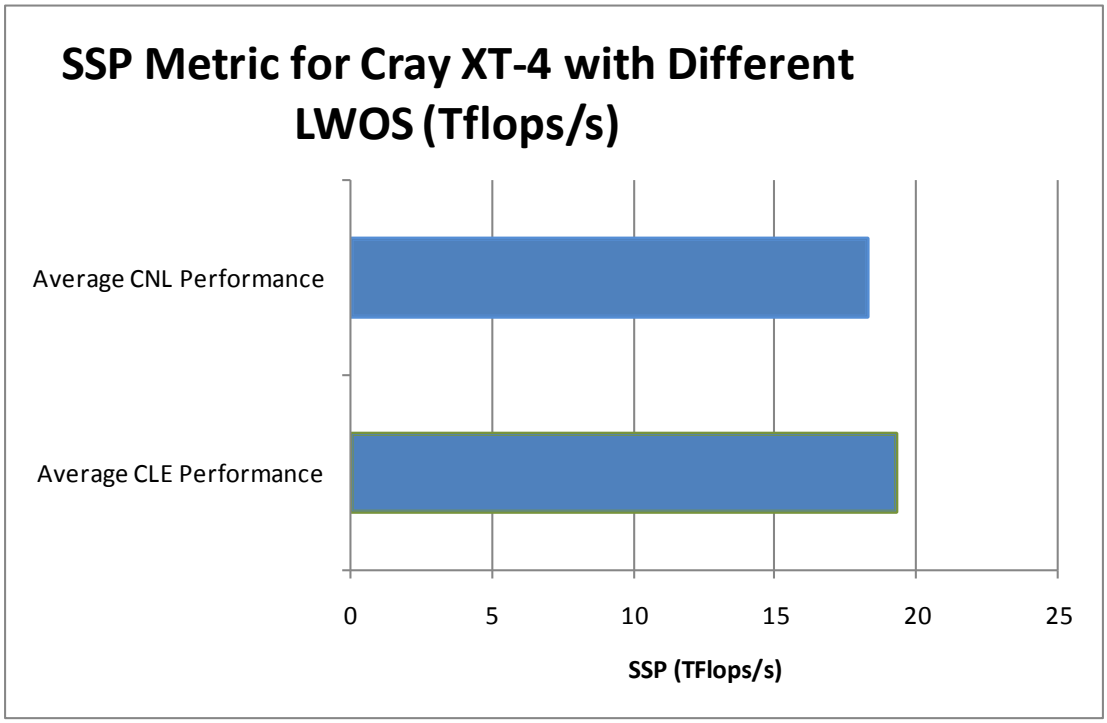


Figure 4-5: The SSP-4 metric for the same XT-4 hardware running two different Light Weight Operating Systems. It was a surprise that CLE outperformed CVN.

Figure 4-5 shows the SSP composite performance for CLE is 5.5% better than CVN, which was surprising. The design goal for CLE, and the expectation was that it would 10% lower performance the CVN. CVN was in operation on multiple systems for several years before the introduction of CLE, and the expectation set by the Cray and others was using Linux as a base for a LWOS would introduce performance degradation while providing increased functionality and flexibility. The fact CLE out performs CVN, both on the majority of the codes and in the composite SSP was a pleasant surprise and helped convince NERSC and other sites to quickly adopt CLE.

4.7.4.4 Comparing Dual and Quad Core Implementations

SSP-4 was used to assess the change in system potency from using dual core processors to using quad core processors instead, with most of the rest of the system remaining intact. The dual core processors were AMD running at 2.6 GHz, each with 2 operations per clock. The quad core implementation used AMD running at 2.3 GHz and with 4 operations per clock. The other change was to change the memory from 667 MHz DDR-2 memory (2 GB per core) to 800 MHz DDR-2.

Comparing the SSP-4 results, the dual core ran at 0.99 MFlops/s per core (NERSC-5-Dual Core, with 19,320 compute processors, which has a system potency of 19.2 TFLops/s) and the quad core 0.98 MFlops/s per core (NERSC-5 Quad Core, 38,640 compute processors, which has a system potency of 37.98 TFLops/s). No special quad core optimizations were done on the codes other than to exploit standard compiler switches. The fact the performance was almost double, despite having a 10% lower clock, was the result of faster memory and the compiler's ability to use the two extra operations per clock.

4.7.5 SSP-5 (2008) – A Sharable SSP Suite

In 2008, the SSP-5 suite was released and became the first suite with complete access to all the tests. Compared to SSP-4, the SSP-5 suite changed two applications, updated versions of other applications, increased

problem sizes, and increased application scale. SSP-5 adds emphasis on strong scaling applications because of the increase of multi-core CPUs.

4.7.5.1 Description of SSP-5

The other major change for SSP-5 is the concept of base and fully optimized cases. Since the same applications and problem cases were used in both, they reflected a general scientific workload.

- The base case can be considered as the way users will initially migrate to a new system. The existing applications base case is designed to represent a system's Potency with a modest effort of porting. In most cases, such porting to move an application, recompile with a reasonable selection of options, and to link in the appropriate system-specific libraries takes a couple of days.

- The fully optimized case, using the same applications and problem cases, was designed to reflect the sustained performance "best possible" case for the application. The fully optimized case can be considered a user spending significant time to restructure algorithms, redistribute the problems, and reprogram the applications for special architectural features. It also allows for code tuning and optimization.

These changes allow SSP-5 to determine how much added potency does a system have, if one were to fully exploit all architectural features to the

maximum amount possible. Most users will take the easiest path – reflected in the base case, but some may spend the effort to better optimize their application. The base and fully optimized cases give the range of expectations for systems.

4.7.5.2 The Base Case

The base case limits the scope of optimization and allowable concurrency to prescribed values. It also limits the parallel programming model to MPI only implementations of the tests. Modifications to the applications are permissible only for limited purposes listed below:

- To enable porting and correct execution on the target platform but changes related to optimization are not permissible.
- There are certain minimal exceptions to using the prescribed base concurrency.
    - ◄ Systems with hardware multithreading
    - ◄ If there is insufficient memory per node to execute the application. In this case, the applications must still solve the same global problem, using the same input files as for the target concurrency when the MPI concurrency is higher than the original target and using the same input files as for the target concurrency when the MPI concurrency is higher than the original target.

- To use library routines as long as they currently exist in a supported set of general or scientific libraries.
- Using source preprocessors, execution profile feedback optimizers, etc. which are allowed as long as they are, or will be, available and supported as part of the compilation system for the full-scale systems.
- Use of only publicly available and documented compiler switches shall be used.

4.7.5.3 The Fully Optimized Case

In the fully optimized case, it is possible to optimize the source code for data layout and alignment or to enable specific hardware or software features. Some of the features the fully optimized case anticipated include:

- Using hybrid OpenMP (Dagum and Menon 1988) and MPI programming for concurrency.
- Using vendor-specific hardware features to accelerate code.
- Running the benchmarks at a higher or lower concurrency than the targets.
- Running at the same concurrency as the targets but in an "unpacked" mode of not using every processor in a node.
    - ◄ When running in an unpacked mode, the number of tasks used in the SSP calculation for that application must be

calculated using the total number of processors blocked from other use.

In the fully optimized case, changes to the parallel algorithms are also permitted as long as the full capabilities of the code are maintained; the code can still pass validation tests; and the underlying purpose of the benchmark is not compromised. Any changes to the code may be made so long as the following conditions are met:

- The simulation parameters such as grid size, number of particles, etc., should not be changed.
- The optimized code execution still results in correct numerical results.
- Any code optimizations must be available to the general user community, either through a system library or a well-documented explanation of code improvements.
- Any library routines used must currently exist in a supported set of general or scientific libraries, or must be in such a set when the system is delivered, and must not specialize or limit the applicability of the benchmark code nor violate the measurement goals of the particular benchmark code.

- Source preprocessors, execution profile feedback optimizers, etc. are allowed as long as they are, available and supported as part of the compilation system for the full-scale systems.

- Only publicly available and documented compiler switches shall be used.

The same code optimizations must be made for all runs of a benchmark at different scales. For example, one set of code optimizations may not be made for the smaller concurrency while a different set of optimizations are made for the larger concurrency. Any specific code changes and the runtime configurations used must be clearly documented with a complete audit trail and supporting documentation identified.

4.7.5.4 Assessment of SSP-5

SSP-5 was released in August 2008. As of this writing, it is too early assess its effectiveness. It was created based on the experiences with SSP-4 and recognizing the increase in the potential for system to have accelerators, multi-cores and special architectural features that will not be exploited without code modification. Preliminary results include runs on several architectures including the Cray XT-4, the IBM Power-5 and IBM Blue Gene and several commodity clusters.

4.7.5.5 SSP-5 Results for NERSC-5

The first system to use the SSP-5 is NERSC's Cray XT-4. On that system, the base case provided 13.5 TFlops/s on the dual core system, and 26 TFlops/s on the quad core system.

The code and benchmark rules can be downloaded from the current NERSC-6 web site, http://www.nersc.gov/projects/procurements/NERSC6 and http://www.nersc.gov/projects/ssp.php.

4.8 Experiences and Impact of SSP

The impact of the SSP methodologies can be seen in a number of ways as described in the following examples.

4.8.1 Revisiting the Real World, Once Removed Example

In the example from Section 3-2, determining the system with the best value, from the individual data was not clear. Using Equation 3-5, a single overall system wide potency measure was obtained for each system. The potency measure was compared with price cost to yield an overall price performance measure – as shown in Equation 3-6. The assessment period was 36 months and $\Phi(\mathbf{W}, \mathbf{P})$ continued to use the arithmetic mean.

Table 4-4 shows the integrated system-wide potency for all the systems in the example, and the relative price performance. Recall the price of each system is proprietary as well as the specific details of the configuration. Hence the cost data is relative to the lowest cost system. Assuming all other

factors (effectiveness, reliability, consistency, and utility) are equivalent, System 3 has the best overall price performance, followed by System 4.

		System 1	System 2	System 3	System 4	System 5
Phase 1 System SSP - Arithmetic Mean	GFlops/s	544.5	360.5	993.1		311.4
Phase 2 System SSP - Arithmetic Mean	GFlops/s	1,089	511	2,731	1,896	1,246
Potency - Arithmetic Mean	GFlops/s* Months* PFlops†	31,037 80,448	15,081 39,090	61,077 158,312	62,570 162,648	39,236 101,699
Average Potency- Arithmetic Mean	GFlops/s* Months PFlops	862 2,235	419 1,085	1,697 4,398	1,738 4,518	1,090 2,825
Potency for relative cost - Arithmetic Mean using normalized cost	GFlops/s* Months per cost unit	661	403	2,156	1,635	1,246

Table 4-4: The Potency and average SSP over time, using the arithmetic mean as $\Phi(W, P)$ and 36 months as the performance period.

4.8.2 Risks of Using Peak Performance as a Selection Criteria

There are many reasons why decisions should not be made based on peak performance, or even benchmarks that closely correlate with peak performance. SSP-1 was used to evaluate and manage the NERSC-3

* Assumes all months are 30 days
† Assumes a month has 2,592,000 seconds.

system. Watching the system evolve is an excellent example of SSP being used not only to evaluate and select systems but also to assess on-going performance, the third purpose of benchmarking.

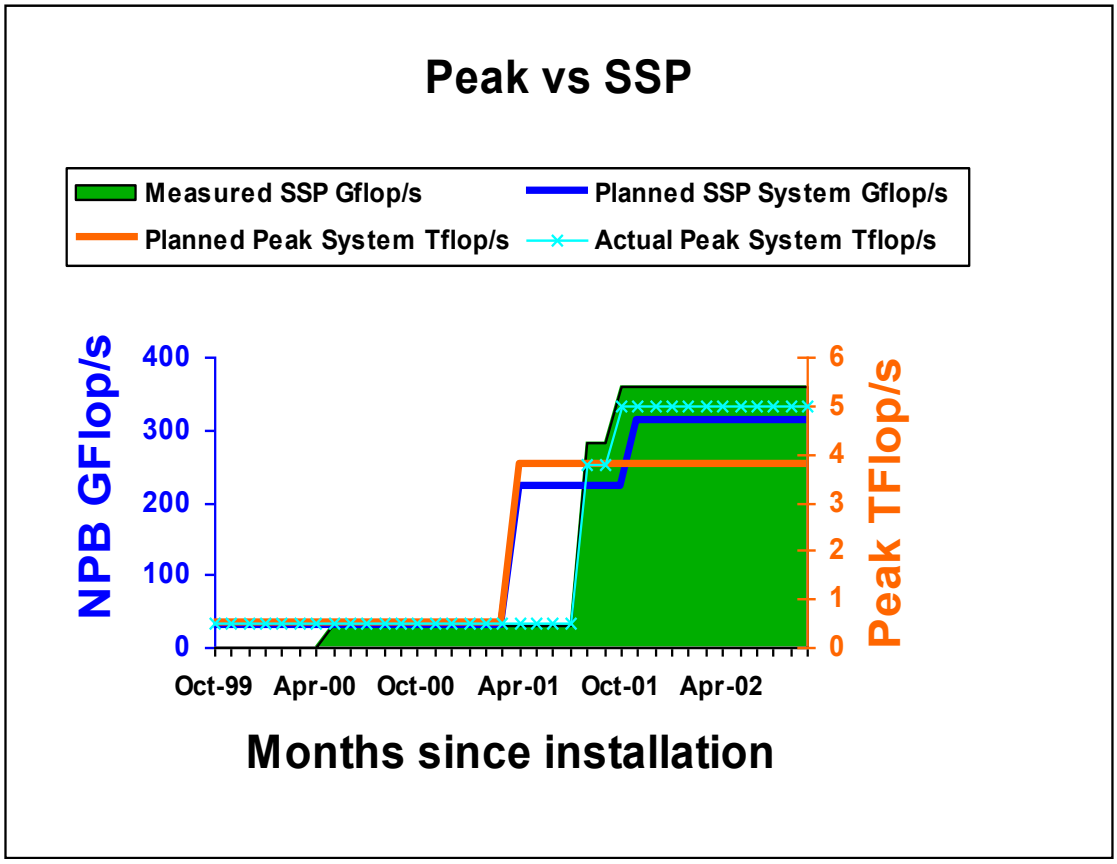


Figure 4-6: Peak vs. Measured SSP-1 performance

The NERSC-3 system was delivered in two major phases – with the first phase consisting of IBM Power 3 (“winterhawk") CPUs and a TBMX (Bender, et al. 1997) interconnect originally planned from Oct 1999 to April 2001. The second phase, planned to start in April 2001, was one of the first systems with the Power 3+ (“nighthawk”) CPUs (Amos, et al. n.d.) connected with the IBM ”colony” High Performance Switch (Lascu, et al. 2003) (Govindaraju, et al. 2005). The schedule and performance levels were agreed upon so the

average SSP performance was 155 GFlops/s over the first 36 months of the contract. In Figure 4-6, the expected SSP of the two phases is shown as the dark line. The Phase 1 system was to start production service in Oct 1999 with an SSP of more than 40 GFlops/s, and it would be replaced with a Phase 2 system in April 2001 at 238 GFlops/s. Phase 2 actually had two sub phases, *a* and *b*. The difference between the two sub phases was strictly software improvements. All the phase 2 hardware was deployed at the beginning of Phase 2a so the Phase 2a and 2b systems had the same peak performance of 3.5 TFlops/s.

The full potency of performance could not be realized at initial Phase 2 delivery. The hardware configuration of Phase 2 was a number of 16 CPU SMP nodes, each with two Colony interconnect adaptors. The Colony HPS switch was the first IBM interconnect that allowed multiple adaptors in a node. The software to use more than one adaptor was only planned to be available eight months after the hardware delivery. The system performance of Phase 2a was limited by the interconnect bandwidth. NERSC-3 showed significantly increased performance once the second adaptor was usable, eventually reaching 365 GFlops/s as the overall SSP. Clearly, the peak performance parameters of the system did not reflect the actually performance potency of the system until the software allowed full use of the interconnect performance.

Figure 4-6 shows another valuable aspect of SSP. Contractually, the Potency starts accumulating only after system acceptance, which occurred

later than expected for Phase 1 and Phase 2. This was due to various issues, including delays in manufacturing, parts availability and software problems that prevented reliable use of the system. The acceptance of the Phase 1 system occurred in April 2000 rather than October 1999 and the acceptance of the Phase 2a system was July 2001 rather than April 2001. On the other hand, the software to exploit the second adapter was delivered earlier than the eight-month delay originally expected, arriving in October 2001. The early delivery of the dual plane software provided a measured performance improvement earlier than planned and partially offset the other delays. The system configuration was adjusted after Phase 2b acceptance so the average of the SSP did meet the required 36-month average.

The SSP method gives both the purchaser and the supplier protection. The supplier has the freedom to adjust the schedule of deliverables and the purchaser is protected by a guarantee of agreed-upon amount of performance delivered in a certain time period. The degree of adjustments can be constrained as well. For example, a purchaser probably does not want all the performance delivered in the last month of the 3-year period, so there may be limits on when the phases are delivered.

4.9 SSP as an On-Going Measure

As mentioned above, benchmarks can be on-going assessments of systems to help assure systems continue to operate as expected. In order to be effective as an on-going performance assessment, benchmarks must be

run often enough to provide enough samples for trend analysis in an on-going assessment*. Running the benchmarks as a regression test after system changes is one approach – but is limited in the number of observations that can be obtained. This also means there is little context to judge significant changes versus random errors. An improvement is running benchmarks regularly – at least weekly to have an understanding that the system performs properly. Further, the benchmarks should run alongside the regular workload as a production job rather than in special circumstances such as on a dedicated system. This allows a better assessment of what the user community sees for performance, especially if the benchmark suite is drawn from the application workload itself. In order to be effective in assessing on-going performance, the benchmark suite should:

1. Be reflective of the workload itself
2. Be able to run within the normal scheduling parameters of the system
3. Run while the normal workload is running
4. Balance running long enough to assess system performance but short enough not to consume undue amounts of system resource

* In fact, they need to run often to provide enough samples for variation analysis as discussed in Chapter 8.

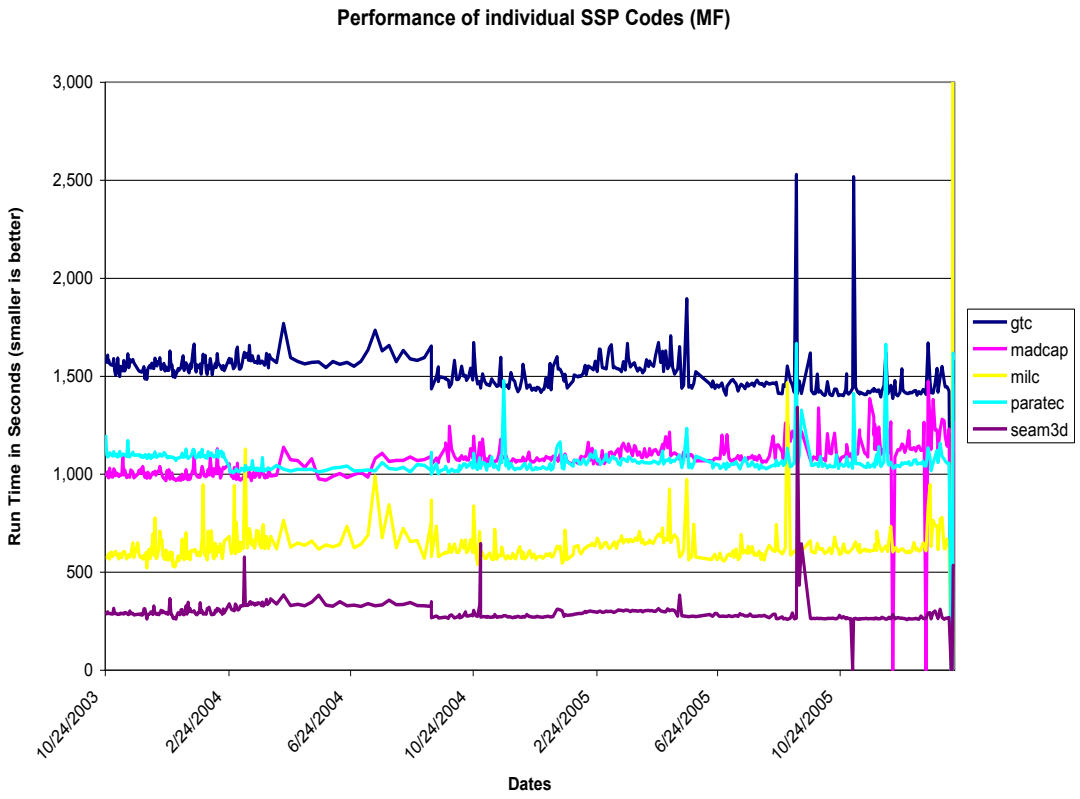


Figure 4-7: Runtimes of the SSP-2 component benchmarks over an extended time.

Figure 4-7 shows the runtimes of the SSP-2 components over two years on the NERSC-3/3E. The runs were done several times a week and ran as standard production jobs. Several insights are notable. First, the runtimes are consistent, with a few exceptions. Note the large spike in runtime. For the most part, the runtimes are within the expected variation of non-dedicated systems. At several points, several of the codes take longer to run, indicating something on the system may be had a detrimental impact on performance, particularly since they appear to be clustered in approximately the same time period. The large spikes are an indicator possibly something is amiss. Once the trend is noticed, further investigation is probably needed to determine

whether there is a system problem. This data can also be used to judge the consistency of the system that is discussed in detail in Chapter 7.

Figure 4-8 indicates the composite SSP-2 values over time. The flat line is the required contractual performance negotiated. The graph shows that several times the actual performance was less than expected. This indicated system issues that were then corrected to return the system to its expected performance levels.

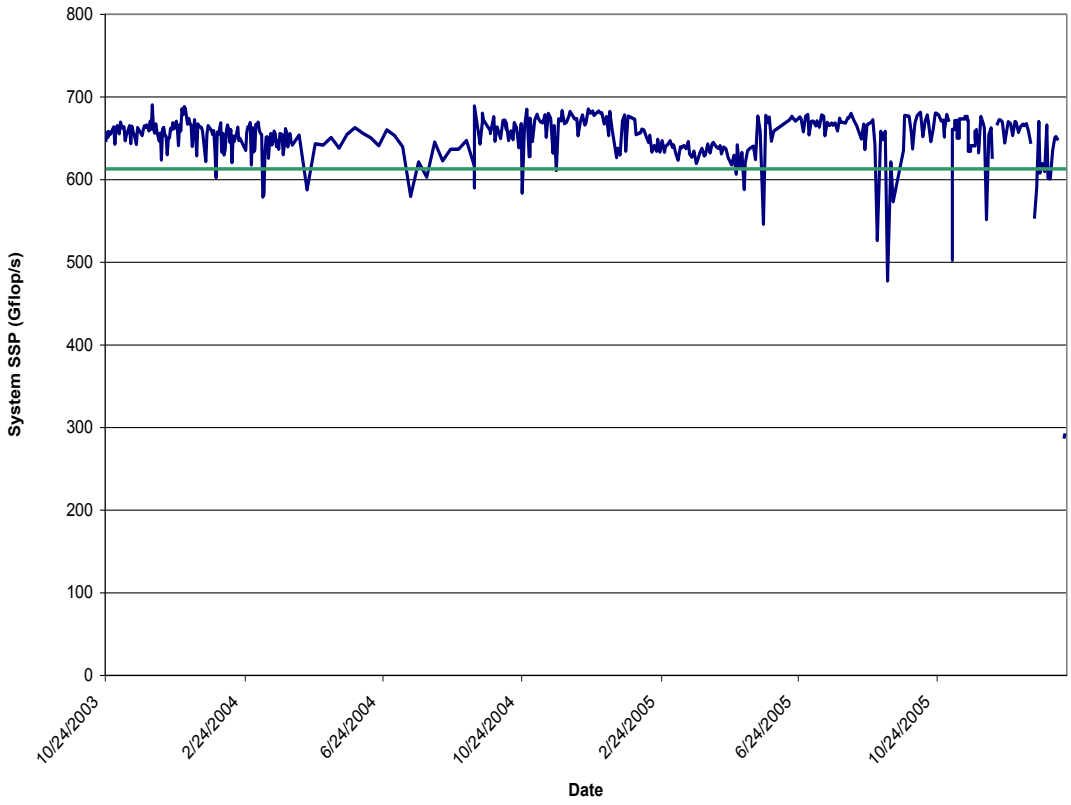


Figure 4-8: SSP validated performance on-going performance of the IBM Power 3 system using SSP-2. The line slightly above 600 is the contract required metric.

The NERSC-3, Phase 1 system had a persistent degradation of performance, measured both by SSP-1 and user applications. The system

slowed down by approximately 5% every month until it was fully rebooted, a process that took close to 3 hours. A reboot would return the system to the expected performance level. This improper behavior was only detected because of proactively running the SSP-1 benchmarks. Since the system was in place less than 18 months and it took time to detect the pattern of gradual loss in performance, it was not possible to definitively determine the cause of the slowdown before the system was replaced with the Phase 2 system. However, recognizing the degradation meant a work around of rebooting the system every month was worthwhile.

4.10 Validating SSP with User Reported Performance

One important question is, “Does the SSP metric reflect the performance of the actual workload that runs on a given system?” This is determined by the careful selection of the component codes that make up the SPP. Once a year, all the projects at NERSC submit a proposal for computer time allocation. Since 2004, every project was required to provide performance data on the application codes they use. The performance data from running jobs is at scale. A NERSC staff member, Dr. David Skinner, implemented the Integrated Performance Monitor (IPM) tool, to assist users profiling their applications and collecting the required data. IPM (Oliker, Borrill, et al. 2005) provides an integrated method to collect data from hardware performance counters.

The 2006 proposal submissions for NERSC allocations were reviewed to determine the characteristics of the workload, showing how well the applications ran during 2004/2005. There were 813 separate performance submissions from 316 different project submissions that had used NERSC systems in the previous period. A performance submission is considered unique if the combined project ID, code name, and concurrency are unique. There were 279 unique submissions reported. NERSC supports a wide range of science disciplines, and the code submissions reflect that as well. Table 4-5 compares the amount of time used and the number of performance data submissions of performance data for applications, both by science discipline. The percentages are aligned, but not exactly because each science area has different numbers of applications and projects. The main point of the comparison is that the performance data covers roughly the same areas as the usage profiles. Since the SSP approximates the actual application performance, even with somewhat of an underestimation, it is reasonable to assume the SSP reflects the workload.

Science Area	Percent of Computational Usage in Allocation Year (AY) 2005	Percent of Performance Submissions based on Data from AY 2005
Accelerator Physics	5%	7%
Applied Mathematics, Mathematics and Computer Science	4%	4%
Astrophysics	12 %	7%
Chemistry	13%	12%
Climate and Environmental Science	8%	7%
Engineering	5%	1%
Fusion Energy	29%	20%
Geosciences	2 %	2%
Life Science	8%	4%
Materials Science	9%	30%
Quantum ChromoDynamics (QCD)	8%	3%

Table 4-5: The table comparing the amount of time used by science discipline and the number of performance data submissions of performance data for applications. The percentages are aligned but not exactly because each science area has different numbers of applications and projects. The main point of the comparison is that the performance date covers roughly the same areas as the usage profiles.

The performance data submissions and the amount of time used by science disciplines are consistent. The science areas with the larger usage are also the science areas with more performance data. Therefore, it is reasonable to use the performance submissions to make general observations of the overall workload, as is done in Table 4-6.

Data Characteristic	Amount
Number of Different Projects	316
Number of Different Submissions	813
Number of Different Submissions based on Seaborg runs	720
Unique Codes	279
Minimum Concurrency	1
Maximum Concurrency	32,768
Seaborg average reported per processor performance	191 MFlops/s
SSP-1 per CPU performance	115 MFlops/s
SSP-2 per CPU performance	214 MFlops/s

Table 4-6: Summary of performance data reported by science projects running at NERSC.

Table 4-6 compares the SSP and measured performance for the applications on NERSC's most parallel system at the time, *Seaborg*, which is a 6,756 processor SP-3+. The result of using SSP-1 is an average SSP value of 115 MFlops/s per processor on *Seaborg* and the next generation, SSP-2 is 214 MFlops/s per processor. The average actual performance reported by the user community for the yearly allocation submission process was 191 MFlops/s per processor approximately three years after the system was initially deployed. The actual aggregate performance of the workload for real production was bracketed by the two SSP values. The first value, actually used to select the *Seaborg* system, was overly conservative and under-estimated the true overall performance. This is to be expected since the reporting of the workload was done well after the system was in service and the user applications had been optimized for the system. Furthermore, the system software, compliers and libraries had improved by that time as well. The next generation, SSP-2, was selected to evaluate the next generation system, was more aggressive in scaling and used codes that had good

performance characteristics. So it is natural to expect SSP-2 to slightly over estimate the average workload (by about 12%).

Figure 4-9 shows the number of performance profile submission by discipline area, along with average per processors performance that were reported.

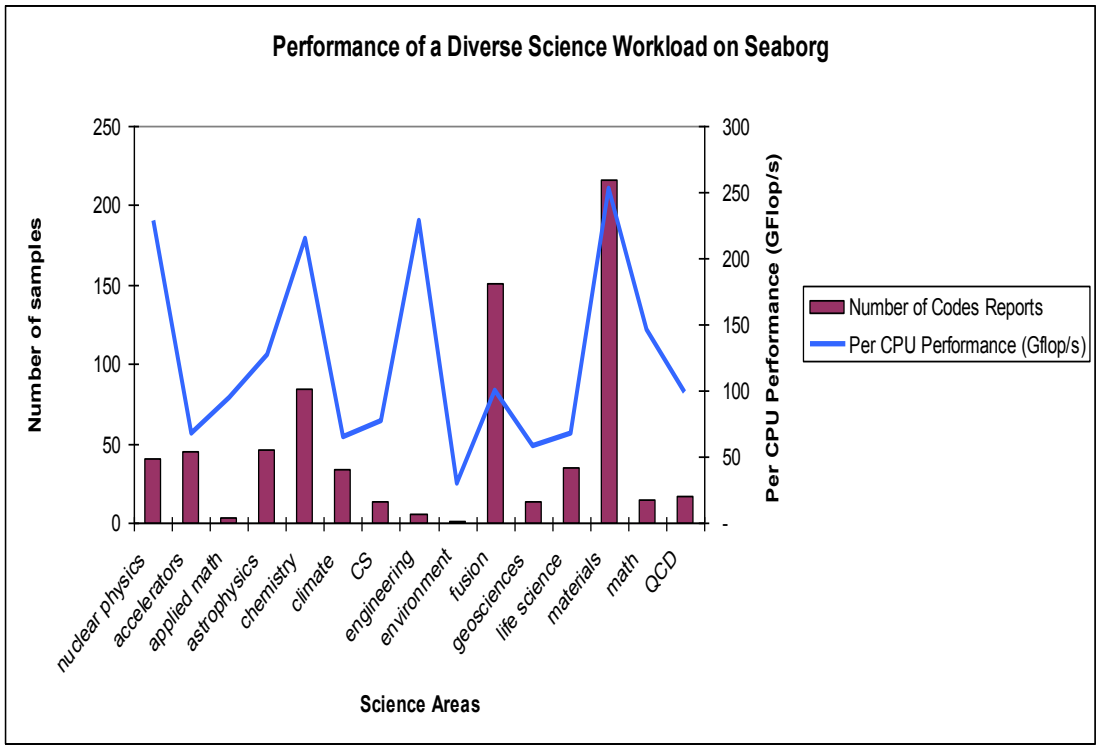


Figure 4-9: Collected hardware performance data for science discipline areas and the CPU Performance data measured using IPM for over 270 applications.

Comparing the profile data from the NERSC user community with the SSP-1 and SSP-2 measures indicated the SSP method is a valid estimate of a systems performance for at least the NERSC workload.

4.11 Observations on Application Modeling and SSP

There is a desire to simplify performance measurement to use simple, low level, synthetic tests (*simple metrics*) because simple metrics are easily

ported and require less system and human resources to run. The more aggressive desire is to only use a limited number of simple metrics to gather system data. It is hoped that results from the simple metrics would then either be used directly or combined with performance models of applications and/or systems to predict full application performance on the system under evaluation with enough accuracy that no full applications tests would be needed.

There are two major thrusts in simple metric performance modeling, *analytical models* of applications and *application instrumentation combined with system models*. The former thrust is represented by the work at the Performance and Architecture Laboratory (PAL) (PAL n.d.) at Los Alamos National Laboratory led by Dr. Adolfy Hoisie and the latter thrust is represented by the work of the Performance Modeling and Characterization (PMaC) Group at UC San Diego led by Dr. Allan Snavely (PMaC n.d.). Both thrusts have claimed success to a degree, so, in conjunction with the NERSC-5 acquisition, the author approached both groups to try to use the respective modeling techniques to predict performance of the SSP-4 suite on systems under evaluation. The concept was to use either or both performance prediction methods and compare the predicted performance with the actual performance across a number of systems. It was also hoped that once a system was selected, the modeling could assist in system configuration trade-off decisions.

Due to the labor, other priorities and time required to perform the detailed analysis necessary to create analytical models for all the full SSP-4 applications, the analytical model approach was not feasible in the time-frame necessary, and was not pursued.

The PMaC approach, as it was implemented at the time, was pursued through 2005 and 2006 in collaboration with Dr. Laura Carrington and Dr. Allan Snavely at SDSC and Mr. John Shalf, Dr. Jonathan Carter and Mr. Noel Keen at NERSC. The process and progress of the effort is reported in(Keen 2006). The steps to implement this version of the PMaC method are:

1. Collect application profile data for each application running at full scale, using several different tools. This required multiple executions of the applications and could only be done on a Compaq ES-45 system with Compaq Alpha ev-68 processors.

2. Obtaining MPI call trace data all applications at each scale of target.

3. Obtain cache, memory and other system characteristic data using series of simple tests [CPUbench (Carrington, et al. 2005), Membench (Membench n.d.), etc.] on representative system under consideration

4. Measuring communication latency and bandwidth for target systems.

5. Convolving the application and system information to make a prediction of performance. The PMaC Convolver uses memory traces and compute resource models estimate of how long that application spent in the memory sub-system.

The results of the attempt to correlate SSP-4 results with PMaC predictions required running multiple versions of 7 applications – each at 4 levels of concurrency. The wall clock time of the instrumented applications ranged from 2 to 14 times the time to execute the application without the instrumentation and the data produced was significant. One application, GAMESS, could not be run because the profiling tools only work with MPI, and, while GAMESS can run with MPI, it is very slow and not representative of the performance it typically obtains using LAPI or SHMEM. Data on hardware architecture parameters were obtained for 7 systems that represented all the architectures of interest.

A number of challenges developed in the use of this approach that extended the time and the effort beyond what was expected. These challenges included:

1) Difficulty obtaining accurate cache system data for proposed machines until there was actual production hardware. Data from existing representatives of the architectures could not be extrapolated with sufficient accuracy to the PMaC analysis.

2) The framework was not capable of predicting the performance of systems that are larger than those used to collect the data. (e.g. one cannot use data collected from a 128-way example run on a smaller system to predict performance for much larger concurrencies for the application.). Due to the extend run times of instrumented applications and the fact the systems being evaluated were substantially larger than the test platforms the tools ran on, this was a limitation.
3) The framework does not account for differences in compiler performance on different systems. For example, the code generated by one compiler can be up to 30% faster than code generated by the another compiler on a different system, even when the slower system has faster CPUs. The framework only does memory traces of the code as compiled on a single system, without regard to the differences in pre-fetch strategies emitted by different proposed compilers.
4) The method required running memory pattern and network data gathering tools on all target systems, some of which did not physically exist at the time of evaluation.
5) All applications had to be ported and run on an Alpha processor machine – the only platform then compatible with the tool suite.
6) All applications had to run at full scale multiple times (at least twice for memory tracing and once for communication tracing.)

7) The method required multiple steps. It was difficult to verify that each step was completed properly on its own.
8) Some parameters to be used as input to the convolving steps that were not easily determined.
9) Significant resources were required to collect and process tracing data.

The results seen for NERSC-5 are aligned with the results seen using the similar versions of the tools for the DOD Modernization Program TI-05 evaluation and acquisition (Carrington, et al. 2005), which report:

a) predicting performance based on simple tests are inadequate for predicting or assessing application performance on systems, with LINPACK being the poorest simple test studied;

b) combining simple tests with optimized weights also is inadequate for meaningful application performance prediction;

c) convolving application traces with metrics derived from a specific set of simple tests (the PMaC methodology) can predict performance of applications to about 80% accuracy for the same system (no comment is made for similar but not exactly the same systems); and

d) acquiring the application specific traces is “painful”. It is noted PMAC has since abandoned this version of their framework and has moved to a new framework for performance prediction (Tikir, et al. 2007)

which recently reported predictions with 90% accuracy on a limited set of applications (Dongarra n.d.) for the same system as was instrumented.

The SSP framework is compatible with using modeled performance prediction instead of actual application runs. However, combining the DODmod results with the NERSC experience indicates performance prediction with simple tests is not sufficient to confidently compare large scale systems for the foreseeable future. It is often the case the difference in cost for systems under consideration is less than the 10%. The 10% accuracy discussed above is based on predications that actually tested the systems targeted. There is less understanding of accuracy for systems that are only represented with previous versions of an architectural family, but only of the application and prediction on the same hardware. The uncertainty increases, and indeed it may not be possible, to predict performance of an application on a new system based on traces and system characteristic information from an earlier implementation of the architecture.

The effort involved with tracing applications is significant, and in the NERSC experience, exceed that to just port and run the actual applications on the overall systems. Furthermore, even if the modeling were highly accurate, and the effort to gather all the data tractable, the approach does not eliminate the need for full application benchmarks because the modeled performance cannot serve the second and third purposes for tests and

benchmarks discussed in Section 3.4 above. Finally, the loss of information captured by simple tests generates increased risk in the acquisition, and integration steps as the system scale increase.

4.12 Chapter Conclusion

This chapter demonstrates different ways the SSP method can be used to assess and evaluate systems. This method provides the ability to define sustained performance estimates based on time-to-solution that are correlated with the actual workload on the system, yet use many fewer applications. The SSP method has been refined several times, and with each refinement, the version of the SSP is explained and assessed. The SSP method is shown to be usable to assure continued system performance.

The SSP method provides a good overall expectation of potency and value for a system. It also is attractive to vendors who prefer composite tests instead of discrete tests. SSP provides realistic assessments of the potency and value of HPC systems.

Chapter 5: Effectiveness of Resource Use and Work Scheduling

5.1 Chapter Summary

Performance is only one aspect of having a system that is productive for its intended client community. The ability for the clients to effectively access the performance when they need it is also necessary. This is the second component of the PERCU methodology and includes a testing mechanism to measure effectiveness.

This chapter deals with methods to assess how effective a system is in providing access to the performance it is measured to have. Major systems are seldom dedicated to single uses, and more often than not, they are used in different modes at different times. The Effective System Performance (ESP) Test is designed to mimic the types of activities and scheduling needs that reflects changing priorities for scheduling and system usage.

This chapter introduces ESP, explains how it can be used and why it has evolved and assesses ESP's uses and impacts. Appendix F provides similar assessment of an earlier version of ESP and Appendix G discusses an evaluation of multiple job scheduling packages using ESP. Specifically, the ESP metric was designed as an incentive to introduce features that help improve utilization of the Cray T3E by 25% and the later the IBM SP 3+ by

10+% at NERSC. ESP was further able to compare different job scheduling software on the same hardware implementation, giving a quantitative evaluation as well as useful feedback to the suppliers of the software systems.

On the Cray XT-4, which was tested in the fall of 2007, ESP was used to evaluate the job scheduling system for both the Cray Virtual Node (CVN) and the Cray Linux Environments (CLE) system software – both running the Torque job management system with the Moab scheduler. Multiple ESP tests were performed in order to guide the adjustment of scheduling parameters. The improvements in system effectiveness rating ranged up to more than 22% based on improvements suggested by ESP.

Before delving into the details of the goals, design, implementations and evolution of the Effective System Performance, it is useful briefly explore the impact the ESP test method can have. Without going into details of the ESP construction, the next section explores the impact ESP-2 had on the Cray XT-4 system that was installed in 2007 as NERSC-5.

5.2 How ESP Helped Improve the NERSC-5 Cray XT-4

A complete explanation of the design and use of ESP is discussed later in Sections 5.8 and 5.9, and the distribution of the jobs run-times and concurrencies are shown in Table 5-1, ranging from 3.25% to 50% of the maximum number of computational elements in the system. Before examining

the goals and details ESP, it is useful context to see one use of ESP that led to specific system improvements.

For the time bearing, the reader can be content to assume the job mix of ESP completely subscribes the system with jobs that are use differing concurrency, run for different amounts of time and require different scheduling parameters. For all but two of the jobs, the scheduler knows only the job concurrencies and the requested job run-time. Two jobs have specific completion parameters the scheduler needs to accommodate. However, if the reader prefers, come back to this example after reading the sections on the ESP design and implementation.

Now, here is an example of using ESP to assess and improve a system's resource management configuration. ESP was used to evaluate the XT-4's capability under both CVN and CLE to effectively schedule work during system evaluation in the fall of 2007. As discussed in detail below, ESP-2 runs a set of 230 jobs of different scale and duration, including two Z tests that use the entire system. This discussion focuses on using ESP-2 to improve scheduling under CLE. Once the job scheduler, launcher and resource manager were functional, ESP-2 was used to tune and improve the software components. Figure 5-1 shows a chart of one of the earlier ESP-2 runs, which took 14,882 seconds to complete. The target time was 13,671 seconds - reflecting about a 75% ESP rating. In these charts, created by Ms.

Sarah Anderson working with Dr. Joseph Glenski and his team* at Cray, Inc. the shades are matched to the job sequence number in the ESP-2 test.

The X axis is the number of CPUs used in the systems (19,320 compute processors). The Y axis is time from the start of the test – with 0 at the top. Hence a job with a concurrency of 1,024 tasks and 1,000 second long is represented by a rectangle 1,024 points wide and 1,000 seconds long. The tick marks on the Y axis are in intervals of 1,000 seconds. The target test time of 13,671 seconds is indicated in the third dashed line. The first and second dashed lines are Z test submission times and are explained below in detail.

The test run in Figure 5-1 shows a large amount of white area – indicating long periods where many processors were idle, which of course, lowers effectiveness. It can be observed that the system started many large scale (many task) jobs early in the test, and deferred the longer running jobs.

* Frithjov Iversen, John Metzner, Sarah Anderson, Kevin Thomas, Woo-Sun Yang, Steve Luzmoor – all of Cray, Inc. and Scott Jackson of Cluster Resources, Inc.

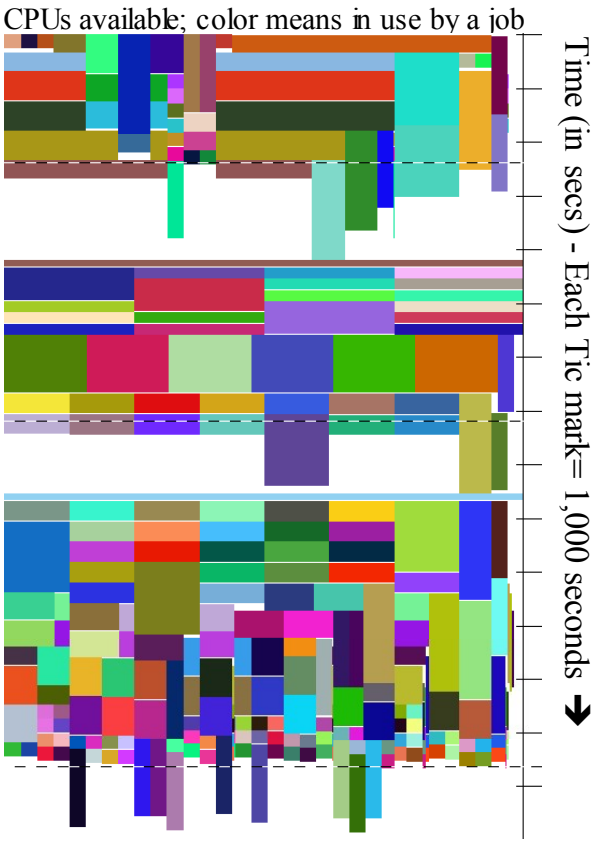


Figure 5-1: An ESP run on NERSC's Cray XT-4.

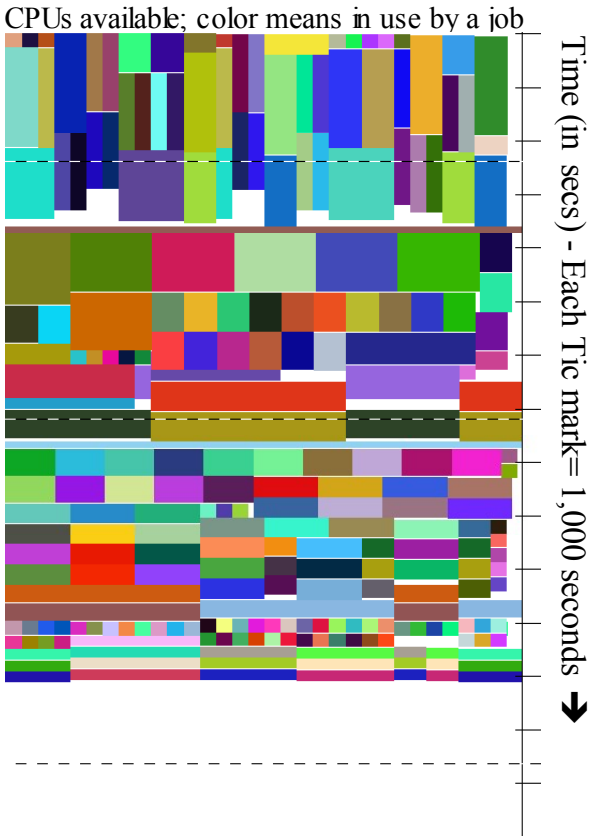


Figure 5-2: ESP run with priority placed on longer running and larger jobs. This test confirms the system can be effectively scheduled and was significantly faster than the target time.

Compare Figure 5-1 with another test represented by Figure 5-2. In Figure 5.2, the test ran in 12,156 seconds – 22% faster, improving the ESP metric. The difference was a better selection of the longer running jobs earlier. Figure 5-2 shows the results of the same test after changing two MOAB parameters. RESWEIGHT was changed from 0 to 1 and WALLTIMEWEIGHT was set to be 1. These changes allowed jobs to be launched in a more deterministic order than the previous selections with the longest job first, if other job characteristics were equal. The result was shortened duration of the drain

periods needed to start the Z jobs. It is also possible to see that the jobs requesting more CPUs were scheduled earlier than in the test in Figure 5-1 and that there were many fewer times with idle CPUs.

The scheduling priority used in Figure 5-2 was aligned with NERSC operational scheduling, which favors the highest concurrency jobs above all others. This policy reflects NERSC's role as a capability system resource and was needed to overcome the tendency of most default scheduling to favor smaller jobs because it was easier for a scheduler to accumulate resources for smaller jobs.

5.3 ESP Introduction and Motivation

The overall value of a high performance computing system depends not only on its raw computational speed but also on system management effectiveness, including job scheduling efficiency, application launch times, and the overhead levels of resource management. Common performance metrics such as the SSP and NAS Parallel Benchmarks may be useful for measuring computational performance for individual jobs, but give little or no insight into system-level efficiency.

The Effective System Performance (ESP) test, first discussed in (Wong, et al. 1999), measures system utilization and effectiveness. The primary motivation for ESP is to aid the evaluation of high performance systems as to their capability of serving a client community that spans different applications, areas and usage modes. ESP can be used to monitor the impact of

configuration changes and software upgrades in existing systems. This test evolved to be an incentive for the development of features to improve system-level efficiency in large scale systems.

The concept of ESP developed as an attempt to describe what we euphemistically called *"a day in the life of a HPC system"*. While not universal, it is typical for large systems to have several operational modes that change over a period of a day to a week. The ESP test extends the idea of a throughput benchmark with additional features that mimic a day-to-day supercomputer center operation. It yields an efficiency measurement based on the ratio of the actual elapsed time the test takes relative to the minimum time, assuming perfect efficiency. This ratio is independent of the computational rate, benchmarks, and compiler optimizations. It is relatively independent of the number of processors used, thus permitting comparisons between platforms as well as changed parameters or software within a system.

ESP is different than traditional "throughput tests" in two significant ways: not all work is submitted at once and the use of full configuration jobs with different runtime expectations. For the former, the job mix is submitted in three groups, separated in time. The number of groups and the time period between them can be adjusted if needed. This means the scheduler has to start jobs before knowing the complete workload parameters – exactly what happens in real life. The use of three groups is a compromise between the

continuous job submission from real life and making the test tractable within a time limit and also predictability. The order of jobs is random, determined by a pseudo-random number generator. In reality, however, the seed for the random number generator is often set to the same initial value for a particular evaluation (say if the metric is included in a statement of work as a metric for a contract). The concurrency and runtime of individual jobs is proportional to the scale of the system as shown in Table 5-1: The ESP-2 Job Mix.

The ESP test was designed to provide a quantitative evaluation of parallel systems in those areas not normally covered by traditional benchmarks but which are, nonetheless, important to production usage. There are a myriad of system features and parameters that are potentially important in this regard. As an alternative to assessing and ranking each feature individually, the ESP test is a composite measure that evaluates the system via a single figure of merit, the smallest elapsed time for a representative workload to complete. This metric translates into the amount of productive usage the system should support over its lifespan.

5.4 Mixed Workload Scheduling

It may be useful to briefly describe the typical operation of a HPC system that supports a varied, highly parallel workload, since it is markedly different than the simpler workloads that run on processor farms of shared memory processor and workloads that run on smaller clusters oriented to

development, transactions or interactive experimental data support. ESP is designed to represent the more complex nature of the multi-function system.

The job mix in HPC facilities is dominated by medium to very large scale parallel jobs that run for many hours to many days. These facilities support hundreds to thousands of users for time periods ranging from months to years. Each user or project is in different stages of development of codes, pre-processing, long “production” simulation and analysis runs, amounts of data analysis, and post processing. Much of this is accomplished through submitting resource scheduling systems jobs containing multiple job steps. The scheduling software is expected to select the most effective combination of jobs based on a set of facility determined policies. Most HPC centers use different policies to select classes of jobs using such parameters as expected runtime, numbers of processors needed, amount of memory needed, and number of other jobs the user or project has running and pending.

5.4.1 The User’s View of Fairness in Job Scheduling

User satisfaction is determined by how well the system sets and meets the turnaround of jobs and provides fair access to the computational and data resources of the system. For this purpose, “fair” is defined in several ways such as:

- “Having or exhibiting a disposition that is free of favoritism or bias;
- Just to all parties;

- Being in accordance with relative merit or significance;
- Consistent with rules, logic, or ethics;
- Moderately good;
- Acceptable or satisfactory" (American Heritage Dictionary – on-line n.d.).

All variants are appropriate to a degree, but perhaps an understandable description of "fairness" as it applies to serving the user community is from Dr. Rick Lavoie who says "the definition of fairness has little to do with treating people in an identical manner. The true definition of fairness is 'Fairness means that everyone gets what he or she needs.'" (Lavoie 2005).

Dr. Lavoie's definition is appropriate since HPC users do not expect immediate turn around for their long production runs. Indeed, depending on the scale and requested time of the job request, along with the systems scheduling priorities, jobs may not be expected to run until late at night or even over weekends. HPC users expect to be able to make predictable progress on their work – an implied if not explicit level of service agreement based on the general dynamics of the behavior of the client populations.

A simple example of fairness that often is encountered in HPC facilities follows. Some work jobs that have dependencies requiring submission of one job being tied to the progress of another job. Climate applications are the

prototypical example of this because it is not possible to simulate the climate of year N with having already having completed year N-1. Other clients have work that is completely independent for each job and hence can submit hundred and even thousands of jobs at one time. quantum chromodynamic calculations fit this model.

Being fair to the first client means that once their first job completes, they do not have to wait for tens to hundreds of jobs that were submitted earlier from the second client to complete, before the first client's second job needs to make progress. At the same time, the second client does not expect all their jobs to complete before anything else is run, but does expect to see some of their jobs making progress.

5.4.2 Job Execution Priorities

Most systems of this type also provide a form of priority so clients can enhance the likelihood of a particular job starting execution. Sometimes this is automatic, in the sense that some projects or classes of jobs[*] are determined to be eligible for a "boost" of priority. It is often the case the job parameter policies change multiple times a day[†]. Other times, users may specify the priority if they have a deadline or need a result[‡].

---

[*] An example of this the NERSC policy announcement to its users on January 11, 2008 for the Franklin system that said "All jobs run in the regular submit class on Franklin that use 1,209 or more nodes will be discounted by 50%."

[†] For example, there may be much more resources devoted to debugging modest size runs during the work day and then a shift to prefer longer running and/or larger jobs outside of normal work hours.

[‡] The description for the NERSC mechanism for this is found at https://www.nersc.gov/nusers/accounts/charging/mpp-charging.php. It says:

"Three classes of batch scheduling are available:

Therefore, imagine adding a third client who has a job that takes the entire resource for a period of time. The third client does not expect to block all other work on the system during the daytime when others are doing debugging, code development and analysis. The third client probably does expect that a job that uses so much resource should run in the middle of the night – or on a weekend – again looking for their work to make progress in some predicable manner. Finally, all the clients are probably willing to expect, at certain times, there is a fourth “high” priority client who has work that has to be done in by a certain deadline (e.g. a weather forecast), and whose work may preempt other work on the system, as long as this does not happen very often.

Of course, each of the clients attends to conferences and may even take a vacation – all at different times – so they are not always submitting their jobs. At any point, new jobs may be submitted that change the scheduling decisions and the relative priority of jobs, so the scheduler does not have complete information, only snapshots of information. Now, multiply this

---

* Premium
* Regular
* Low

A premium job is scheduled for execution before an otherwise equivalent regular job. A low job has a lower priority for scheduling. These priority classes affect how quickly a job is scheduled for execution in the "wait queues"; it does not affect the UNIX priority at which the job executes.

Charging for Priority Batch Scheduling Classes - Priority scheduling classes have different charge rates. It is intended that most users, over the year, will run most of their jobs in the regular class. Users should use the premium class with care; no additional allocation is available to cover the extra charges associated with its use.

Rates:

* Premium scheduling at an elevated charge rate (2.0)
* Regular scheduling at the standard charge rate (1.0)
* Low scheduling at a reduced charge rate (0.5)”

example by 100 or 1,000 to imagine the complexity of scheduling an HPC resource in a multi-user center.

Because of priority processing, (different scheduling policy based on job parameters and deadlines) jobs are seldom scheduled on a strictly first come, first served basis. Certain jobs are given priority over others, even if they have not been waiting as long. ESP is designed to estimate a system's ability to support such a complexity of work and expectations.

5.5 ESP Design Goals

The overall design goals for the Effective System Performance test are:

1. Independence from the effects of processor speed or compiler improvements on the test codes so that system management features remain the focus of the test.
2. Ability to assess the potential for a system to support different operational scheduling modes.
3. Scalability and repeatability of the test so it can be used on systems of different concurrency and scale, as well as to compare system improvements over time.
4. Ability to reflect operational paradigm shifts.
5. Ability to reflect the performance of a scheduler as it operates with incomplete information.
6. Ability to evaluate the efficiency of job scheduling and job launching at scale.

7. Ability to encourage new features that improve a system's ability to schedule work effectively.

5.6 Scheduling Large Jobs

5.6.1 Throughput of Large-Scale Jobs

The throughput of large-scale jobs is an on-going concern at large HPC facilities, since without this focus, the rationale for acquiring and operating a large tightly-coupled computer system does not exist. As mentioned above, the ESP test has 230 jobs that range in scale from 3.25% to 50% of the maximum number of computational elements in the system. This is the workload that begins the test and represents a possible mix of jobs on parallel systems.

The ESP test also includes two "full configuration jobs" that use all the computational resources of a system in one job with concurrencies equal to the total number of available computational cores. These, for no specific reason, are called Z-jobs in the test scripts. The run rules for the ESP test specify that full configuration jobs cannot run at the beginning or end of the test period. This is because in real life, systems have a continuous flow of jobs and often there is not a start or stop period.

The first full configuration job is only to be submitted after 10% of the estimated minimum test time has elapsed such that it is non-trivial to schedule since a workload is already running. The first test has to run before

any other work is started. Similarly, the second full configuration job must complete within 90% of the test and is not simply the last job to be launched. The requirement to run these two full configuration jobs is a difficult test for a scheduler, but it is nonetheless a common scenario in capability environments.

5.6.2 Considerations for Executing the ESP Jobs

Large systems typically require system administration to maintain and improve their operation. These activities require system outages, either scheduled or unscheduled, and the time required for shutting down and restarting the system. Each of these considerations can significantly impact the overall system utilization. For these reasons, the ESP-1 test originally included a shutdown-reboot cycle, which was required to start immediately after the completion of the first full configuration job. However, this was later removed in lieu of specific system management features and functional tests. The utilization efficiency can be computed as follows.

Definition	**Explanation**	**Units Generic [Used in this paper]**
E_s	This is the effectiveness ratio of system *s*.	Percent [percent]
χ_i	The concurrency of test code *i*.	[Processors]
$\mathbf{T_i}$	The time for ESP test code *i* runs	Time [seconds]
$\mathbf{T\text{-}BEST_{s,k}}$	The optimal time of the ESP test code runs for system *s* in phase *k*. In the ESP-2 implementation, T-BEST is the sum of work all the jobs do (concurrency * run-time) divided by the system size.	Time [seconds]

$N_{s,k}$	The number of computational processors for all types of processors α, in system s during evaluation period k.	Time [seconds]
$p_{s,k,\alpha,i,j}$	The per processor performance of test code *i* executing data set *j* on processor type *α* on system s during phase *k*.	Ops/(proc*sec) [Flops per second per processor]

$$E_{\mathrm{s,k}} = \frac{\sum_{i=1}^{I}(\chi_i * T_i)}{\left[N_{s,k} * \left(T - BEST_s\right)\right]}$$

Equation 5-1: The Effectiveness ratio is the time the test actually runs compared to the time the best packing solution indicates.

The jobs in the ESP suite are grouped into blocks (denoted as B) and the order of submission is determined from a reproducible pseudo-random sequence so the job submission order is not fixed. While B can be any number, the ESP has been used with B=1 and B=3. In the case of B=3, the total number of CPUs requested in the first block is at least twice the available processors and the number of CPUs in the second block is at least equal to the available processors. The remaining jobs constitute the third block. The first block is submitted at the start, with the second and third blocks submitted 10 and 20 minutes thereafter, respectively. This structure was designed to forestall artificially configured queues specific to this test and, at the same time, provide sufficient queued work to allow flexibility in scheduling. No manual intervention is permitted once the test has been initiated.

5.6.3 Operational Transitions

One aspect of system operation that is not captured in a standard throughput testing is the concept of operational transitions. In most cases, the job mix of systems differs throughout the day.

Many systems support a more interactive or development job mix during the day – running smaller, shorter jobs to support debugging and testing. At other periods (e.g. after standard working hours), more long running jobs, often with higher concurrency are run – the capability workload. In centers with production requirements, deadline processing may occur where one or more large production jobs have to be started by a particular time. The most obvious example of this is weather centers that have to produce a forecast at a particular point in time. At other times, other "research" jobs taking most of the resources and more parallelism may need to be run.

While not required by a specific time, the systems level of service agreements mean that such jobs cannot wait indefinitely. ESP is designed to represent a subset, indeed one of the hard ones, of the operational transitions that are seen in operational centers.

5.7 ESP: A Method That Can Be Applied To Different Systems and Workloads

The steps above are general and the applications used in the ESP can come from any workload. The rest of the discussion of the first version of the test, ESP-1, describes the actual implementation at NERSC, along with the

results of that implementation. Following that, there is a discussion of the second version of ESP that is generalized, more portable and adaptable.

A facility that wants to use the ESP approach can design their own implementation by following these steps.

1. Identify the applications that most reflect the workload being represented. Once identified, select the concurrency and runtime of the applications. Unlike most throughput tests, each application should have input data for several concurrencies and runtimes. The combination of applications and input needs to be sufficient to allow the test to oversubscribe the number of CPUs in the system while running long enough to provide a valid result. The length of the test is also determined by the number, concurrency and runtime of the jobs.

2. Identify the operational paradigms for the facility. Is the same scheduling priority reflected all the time or does it change over time? If the latter, then select one or more codes to use for the different operational paradigms. It may be the site does not do full configuration jobs, but rather provides a service where ½ the system is dedicated to jobs or it may be a site that is required to guarantee certain jobs run within a fixed time. The Z tests, then, are crafted to reflect these operational shifts.

3. The site can decide how many submission blocks to use. If B=1 and no Z tests are run, ESP decomposes to a traditional throughput test.

4. The scheduling parameters are selected. It is best to use those that are expected for real use so the test reflects the true workload.

5. For a given system, T-BEST is determined by dividing the sum of all the job (concurrency * run-time) by the system size. Note T-BEST is simply a convenient definition of a lower- bound. It is not possible to obtain the T_BEST in a system, but the closer to unity it is the better the system is scheduling work.

The next section illustrates how these steps were implemented in an actual test method at NERSC.

5.8 ESP-1 – The First Implementation

The initial implementation of the ESP test – called ESP-1 to distinguish it from the current implementation, is discussed in detail in Appendix F. This test used the application codes from SSP-2 to create a test based on the goals discussed in Section 5.5. While successful, using applications with non-adjustable problem sets proves limiting since it means that each time a system with of a different scale was evaluated, all the problem sets needed to be adjusted to provide an appropriate amount of work for the test. This also meant that it was hard to compare ESP-1 test implementations across

different scale systems. This led to the creation of the ESP-2 test, which is currently in use.

5.9 ESP-2 – A Flexible Test

The Effective System Performance (ESP) test was devised to provide a metric for production-oriented parallel systems that is primarily focused on operating system attributes and has been through two major implementations in 1999 and 2002-2003. Such attributes include parallel launch time, job scheduling and preemptive job launch. The ESP-2 test has been deliberately constructed to be processor-speed independent with low contention for shared resources (e.g. the file system). As such, it is different from a throughput test that is influenced by processors speed and compiler performance and assumes a single operational paradigm. The goals of the ESP-2 are consistent with the ESP-1, but are more specific to measure the scalability, stability and effectiveness of the system scheduling and resource management software. Another goal was to make ESP-2 more portable and easier to use. The ESP-2 approach is to run a fixed number of parallel jobs through a batch scheduler in the minimum elapsed time. Individually, the jobs are designed such that their elapsed runtimes can closely approximate a fixed target runtime. The target run-times are provided to the scheduler as the requested run-time for the job. Thus the elapsed time of the total test is independent of the processor speed and is determined, to a large degree, by the efficiency of the scheduler and the overhead of launching parallel jobs.

In ESP-2, a job is a simple, MPI based, kernel program that does simple computation for the targeted amount of the time. Each job does very modest MPI communication. The simple kernel is sufficient since the goal of ESP jobs is to simply occupy the system resources. The test is not trying to assess the performance of compilers or hardware, in fact just the opposite – it is designed to be independent of these influences.

In ESP-2, there are 230 jobs derived from a list of 14 job types, which can be adjusted if a different proportional job mix is needed. The concurrency of each job run scales with the entire system size in order to keep the test constant relative to the number of cores or CPUs. Table 5-1 shows the job types with their relative size compared to the entire system, instance count and target runtime.

Job Type n	Fraction of Job Concurrency relative to total system size δ	Count of the number of Job Instances κ	Target Runtime (Seconds) π
A	0.03125	75	267
B	0.06250	9	322
C	0.50000	3	534
D	0.25000	3	616
E	0.50000	3	315
F	0.06250	9	1846
G	0.12500	6	1334
H	0.15820	6	1067
I	0.03125	24	1432
J	0.06250	24	725
K	0.09570	15	487
L	0.12500	36	366
M	0.25000	15	187
Z	1.00000	2	~100
Total		230	

Table 5-1: The ESP-2 Job Mix

The fractional *s*ize is simply the concurrency of the job as a fraction of total system size. For example, if the system under test has 1024 cores for computation, then the concurrency of job-type B is 64 (= 0.06250 x 1024) cores. Thus, the ESP-2 test can be applied to any system size and has been verified on 64, 512, 2048, 6726 and 19,320 computational core systems.

For the purposes of this discussion, it is useful to define the ESP unit of computational "work" as the product of the runtime of a job and job concurrency (number of cores). Following our example, job-type B is designated 64 CPU x 322 seconds = 20,608 CPU seconds of work. Therefore, for a 1,024 computational core system, the total amount of work, ω_s in the ESP-2 test is seen in Equation 5-2.

$$\omega_{s,k} = \sum_{n=1}^{14} N_{s,k} * \left(\delta_n * \kappa_n * \pi_n\right)$$

Equation 5-2: The amount of work ESP-2 based on system scale, for a given system s and a point in time k.

For a system with 1,024 processors, and not counting the Z type jobs since their time will slightly vary based on system size, the work is 11,031,792 CPU seconds. Given a total amount of work ω_s, a hypothetical absolute minimum time, (T-BEST), can be computed by dividing the work by the system size. In this case, T-BEST = 10,773 seconds (~ 3 hours). Note that T-BEST is independent of the total system size and the processing speed of the system. The ESP efficiency ratio ε is defined as the T-BEST divided by the

observed elapsed time of the ESP-2 test. This is the key metric of the ESP test. For increasingly efficient systems, the ratio approaches unity.

The T-BEST is simply a convenient definition of a lower bound. It is not possible to obtain the T-BEST in a real test even in the optimal case. Therefore, most attainable ESP-2 ratios fall in the range of 0.6 - 0.8 based on the ESP test runs on NERSC systems. Furthermore, the T-BEST must be computed for each system tested, as the runtimes for each job will only approximate the target runtimes.

The ESP-2 test first requires executing the job mix described by the Table 5-1 excluding the job-type Z (228 jobs). This is designated the “throughput” variant of ESP-2. The order of job submission is determined by a fixed pseudo-random sequence, so scheduling software cannot be preset to assume a job submission flow.

$$\mathcal{E}_{\mathrm{s,k}} = \frac{T - BEST_{s,k}}{\sum_{i=1}^{I} (\chi_i * T_i)}$$

Equation 5-3: The Effectiveness ratio is the time the test actually runs compared to the time the best packing solution indicates.

The second part of the test is identical to the “throughput” variant except that the two Z jobs are submitted at 2400 and 7200 seconds after the start of the test and after all other jobs, A, ..., M have been submitted. This is

designated the “Multimode” variant. The Z jobs must be launched as soon as possible. That is, no other queued job is permitted to start while there is a Z job in the queue. This may be accomplished by assigning the Z jobs high priorities, but on most systems it is not sufficient. Further expediting the Z job will depend on how the system handles running jobs; some options include roll-out, checkpoint or suspension. As a last resort, the schedule can simply drain the system of running jobs until the Z job fits. The ESP-2 test does not mandate how this is achieved, simply that no other job is permitted to start running in the interim between the submission of the Z job and its launch.

The ESP-2 test and detailed instructions for installation are located at http://www.nersc.gov/projects/esp.php.

5.9.1 ESP-2 Experiences

The ESP-2 test runs in the range of 4-6 hours while processing 228 jobs (not the Z jobs) on the IBM SP/3+. This is a contraction in the changing scheduling parameters of an operational day that is the result of a compromise between a throughput test of the scheduler, batch system, resource manager and job launcher, and the practicalities of running the test. The overhead associated with node reservation and parallel launch have a large impact in this test.

The ESP test was designed to provide a quantitative evaluation of parallel systems in those areas not normally covered by traditional benchmarks or

throughput tests but, nonetheless, are important to production usage. There are a myriad of system features and parameters that are potentially important in this regard. As an alternative to assessing and ranking each feature individually, the ESP test evaluates the system via a single figure of merit, the smallest elapsed time of a representative workload. This metric translates into the amount of productive usage of the system over its lifespan.

The ESP-2 test is not a scheduler benchmark per se. However, it is obvious that the choice of scheduling strategy will have a significant effect. At first glance, and borne out in real tests, a backfill scheduler with some form of priority preemption is optimal. Although, the difference between backfill and, say, a FIFO (First In First Out) strategy is not as large as one would initially estimate. This is partially due to the composition of the workload. Observations of day-to-day usage show that the union of backfilling a static queue and priority preemption is one way of balancing the competing requirements of high utilization and responsiveness.

5.10 Additional ESP-2 Results for NERSC-5

Section 5.2 shows results using ESP-2 to evaluate the Cray XT-4. In addition to the discussion about setting scheduler parameters to meet the expected time, it is noteworthy that ESP-2 made other contributions. ESP-2 ran on both CVN and CLE versions of the system software.

For the CVN runs, ESP-2 encountered a minor obstacle since it used standard Linux/Unix system calls within for each task to get the time. Each

task then uses that time returned to calculate how long it should run since jobs are self-terminating. CVN provided no mechanism for a task to get system time or time of day, so a new routine was added to the ESP-2 jobs. This was just a minor inconvenience. More importantly, ESP-2 failed on CVN a number of times. These failures were due mostly to the large number of nodes in use during the test. A variety of hardware and software problems were detected using ESP-2 as a blunt diagnostic.

ESP-2 on CLE also brought to light a number of problems, particularly with the early versions of Torque (a resource scheduling system by Cluster Resources, Inc.), which had just been ported to the CLE environment. CLE used an entirely new resource manager – the Application Level Placement Scheduler, ALPS – which replaced the resource manager on the CVN systems. The interaction between Torque and ALPS needed to be refined. ESP-2 helped identify the length of time it took to start jobs, the load balancing for the job scheduling nodes, and other issues. Furthermore, the ESP-2 workload continued to uncover infant mortality problems with the hardware components.

Thus, ESP-2 was an excellent stress test in its own right, in addition to validating the job scheduling and launch software's effectiveness. Appendix G discusses in detail another use of ESP – the evaluation of different resource management systems on the same hardware base.

5.11 Chapter Conclusion

This chapter described a system utilization benchmark, ESP, which has successfully run on multiple highly parallel supercomputers. This test provides quantitative data on the utilization and scheduling effectiveness to which the systems are capable. ESP also provides useful insights on how to better manage such systems.

The most important conclusion is that certain system functionalities, such as checkpoint/restart, swapping and migration, are critical for the highly efficient operation of large systems. This chapter discusses the evolution of the test – from its first concept of evaluating how well a system can support different operational modes for scheduling – to providing a portable and comparable metric.

ESP was improved and made portable by replacing the site-specific application benchmarks that are not freely distributed with other tests that are completely sharable. ESP-2 was simplified to use three or four test codes rather than the original eight applications in ESP-1.

Another effort is to be able to scale the test to more or less CPUs and still have a comparable set of data. ESP-2 is packaged as freely available software archive, with facilities for simple installation and execution. It is located at http://www.nersc.gov/projects/esp.php. In this way ESP-2 can be used by others and will help spur both industry and research to improve system utilization on future systems.

Chapter 6: **Reliability**

6.1 Chapter Summary

Reliability has been a reactive rather than a proactive consideration for large scale system evaluation. It is reactive because it is assessed after the system is installed and operating, and there is not a good way to assess reliability claims proactively. Hardware components occasionally undergo testing to determine mean time between failure (MTBF) and reliability. Component MTBF is then used to predict equipment reliability. System vendors tout reliability enhancing features they add to hardware such as redundant components and dual paths. These can contribute to improvements, but there is little objective understanding of the value of each addition, including the cost effectiveness of any attribute.

This chapter shows that software reliability is often the “Achilles heel” of large systems. Data from NERSC, documented below, indicated that software is the greatest cause of system wide failures, far greater than hardware on a number of systems. Furthermore, vendors seem unaware of this trend since most do not even track reliability in their software components.

System component count is growing despite the fact components are increasing in transistor counts and function. The move to Open Source increases the software reliability issues because there is less integration and

testing of consistent software stacks. Indeed, it can be said, with few exceptions, that every cluster system is a unique collection of software components. Further, there is no overall design process that might help insure some degree of reliability for much of the software.

This chapter investigates failure data at NERSC and other locations. The NERSC failure data spans five years for some systems and covers all systems. The implication of failure trends is discussed and expectations calibrated. Additionally, reliability and failure analysis being applied to HPC systems is examined and some recent related work by others is assessed for it potential use in HPC system evaluation.

Unfortunately, there is no silver bullet for a simple metric that can proactively assess systems before being placed in operation although, as discussed below, some promising trends can be observed. Traditional stress testing and availability periods are ways most facilities identify issues, but these are not sufficient for good evaluation.

6.2 Analysis of the NERSC Reliability Data

Failure data for all NERSC systems from 2001 to 2006 was assembled from the NERSC operational trouble ticket system in which operations and systems staff record all system outages and issues. In addition to the operational logs, data was accumulated from paper records of repairs kept by operations staff, vendor repair logs, and automatic operating system error logs. The data was assembled for analysis in a *mysql* database. Each data

record was manually reviewed and correlated with other information so the information in the database is as consistent as possible. Redundant and overlapping records were combined. Further, each event was reviewed to determine the most likely subsystem category that generated the error.

The NERSC failure data is available at a web site – http://pdsi.nersc.gov - as part of the Petascale Data Storage Institute SciDAC research collaboration. The web site allows interactive queries, charting and exporting of the data. The NERSC systems covered during this time period were the IBM SP 3+ *Seaborg* (2001), the IBM SP 5 *Bassi* (December 2005), the Linux Networx AMD/IB cluster *Jacquard* (July 2005), the SGI Altix 3200 *DaVinci* (September 2005), the High Performance Storage System *HPSS* (from 2003), the NERSC Global Filesystem, *NGF* (October 2005) and the commodity cluster system *PDSF* (2001). The dates show the beginning of the data collection period in the data base, which corresponds to the date of production or 2001 (the start the data collection).

The *Franklin* Cray XT-4 failure data, which begins in October 2007 is not part of the database, but has been compiled separately and correlated with the data in the data base. This enables comparison of the two largest NERSC systems – Seaborg with 6,756 cores and Franklin with 19,576 cores.

6.3 Software and Hardware Errors

The analysis of the data shows that for five of the six systems, software is the primary cause of down time of the entire system. In some cases, the

amount of time a system is down due to software is more than five times that of hardware generated outages. Figure 6-1 shows this data as the percent of unscheduled downtime for six major NERSC systems. Franklin, Seaborg, Bassi, Jacquard, DaVinci and PDSF are computational systems of various architectures and HPSS is a large data archive.

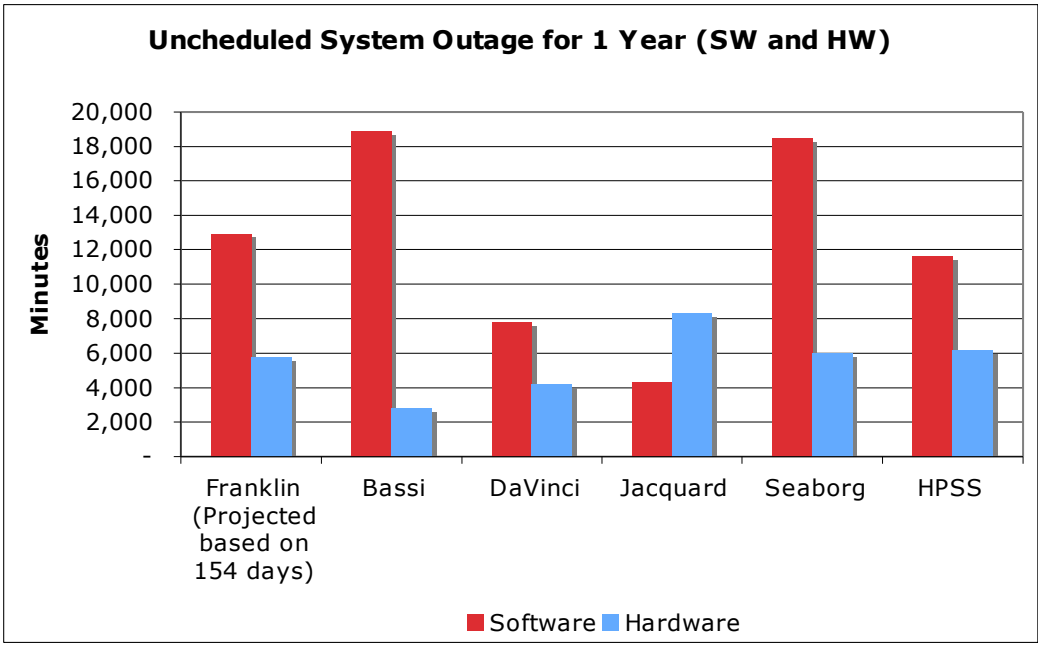


Figure 6-1: Total number of unscheduled downtime for the major NERSC systems over a 1 year period. All systems other than Franklin are from 2006. Franklin is a partial year - 154 days spanning late 2007 and 2008 which is projected to a full year for comparison.

The observation period is for a one year period. The collected data for all systems other than Franklin is for 2006. Franklin had been in service for less than half a year at the time of this analysis, 154 days to be exact, from October 26, 2007 to March 28, 2008. For comparison purposes, the Franklin times are projected to a full year by multiplying by 2.37.

There are several aspects to be considered in the comparison. First, Franklin and Bassi were in their initial period of operation, while the other

systems were in operation for at least year before the data collection period. Hence, it may be expected that the number of outages for Franklin and Bassi will be reduced for later periods. While it is too early to tell about Franklin, this is exactly what happened to Bassi. Comparing the downtime between 2006 and 2007 indicates the hardware downtime decreasing by more than a factor of 2 and the software improving by more than 6 times. However, even with those improvements, software still caused 2.4 times more downtime than hardware.

Only Jacquard shows more hardware downtime than software. This is in part due to a continuing problem with memory components during 2006, which produced very significant hardware downtime. If this set of outages is ignored, Jacquard also shows more software outages than hardware.

The assignment of an outage to the hardware or software category and, in the next section to the subsystems, is not foolproof. Root cause analysis was not done for all failures but instead the most likely cause was assigned. For example, it may be that some software outages had an underlying hardware cause which contributed to the failure but was not reported. The assignment of the failure category is guided by the type of corrective action taken by the system managers and the vendor support personnel.

6.3.1 Subcomponent Error Analysis

Looking more closely at the two largest systems, Seaborg and Franklin, it is useful to compare outage times by subsystem for outages that generated system wide failures. A system wide failure is one where the entire system is unable to meet its Level of Service Agreements. Individual failures occur that may not degrade system performance sufficiently to cause a system wide failure. NERSC uses the following definition for system wide failure.

An entire system is considered down if the system is unable to process work at an agreed upon level. Many components in the system have redundancy such as spare compute nodes and login nodes or alternative routing in an interconnect and for I/O access. A system wide outage occurs if any of the following requirements cannot be met from any part of the system:

- Able to complete a POSIX 'stat' operation on every file within all file systems and access all data blocks associated with these files.
- Able to complete a successful interactive login on at least 75% of the login nodes in the system. (Note: failures in the local area network do not constitute a system-wide failure.)
- Able to run the NERSC benchmark suite for that system, including the full configuration test.
- Able to provide the agreed upon file system bandwidth and all files are accessible.
- Able to make use of the full interconnect bandwidth available. For systems that can route messages in multiple paths, some links or paths may be out of service but only to a negotiated limit.
- All nodes being able to have access to external networks and bandwidth is at least 75% of maximum network I/O node bandwidth.
- Able to support user applications submission, launching and/or completing via the job scheduler.
- Avoiding other failures that reasonably disrupt work on a large portion of the nodes.

- Able to exchange a working spare node when a compute node fails. The number of spare nodes is negotiated based on the total number of compute nodes.

The hardware failure category is separated in three major areas; a) node and interconnect hardware, b) storage hardware and control hardware and c) primary control workstation. Software failures are placed into six categories; a) accounting, b) file system, c) interconnect, d) Internet Protocol (IP) networking (external networks to the system), e) job scheduling, f) security and g) various. The *Various* category covers license servers and mis-configurations, among other things. These categories were those actually used on one system or the other to catalogue trouble tickets. Figures 6-2 and 6-3 show the breakdown by total downtime for Seaborg and Franklin. Seaborg was in service more than 10 times longer than Franklin, so while the scales are the same, the totals cannot be compared.

Comparing the charts indicate the majority of hardware problems are node and interconnect related and not shared storage. Seaborg has local disks, Franklin does not, and local disk failures may sometimes have been recorded as a node failure. Section 6.3.3 explores hardware node failures in more detail. Both systems suffer the majority of their software outages, from either interconnect software or file systems. A word of caution is that symptoms of other subsystem failures, such as interconnects and nodes, can exhibit as file system failures since the file system is spanning all components.

Figure 6-5 shows the two systems together in a normalized comparison of downtime. For the comparison, the amount of downtime was divided by the total time period of the data collection – giving the average minutes of downtime per day. Seaborg has outages in more categories – which may be due to the longer time it was in service. Seaborg also has less average downtime than Franklin because the number of outages per unit time is less, not because the length of each outage was less.

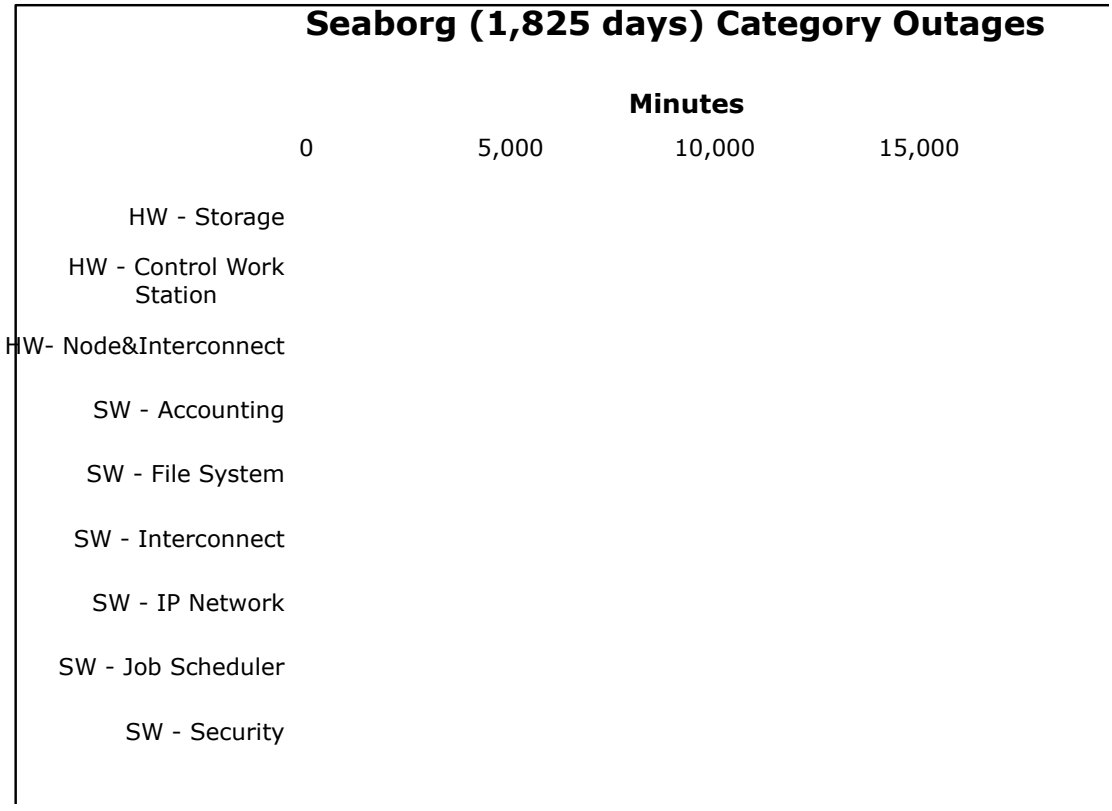


Figure 6-2: Seaborg downtime by hardware and software subsystems.

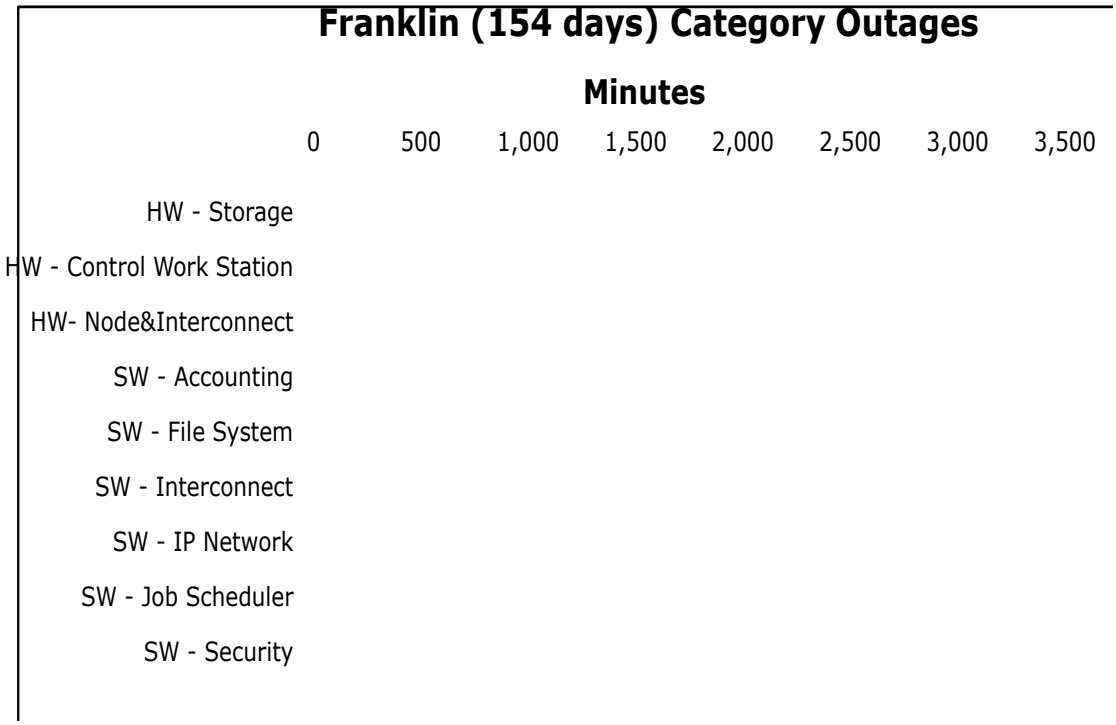


Figure 6-3: Franklin downtime by hardware and software subsystems for a limited time.

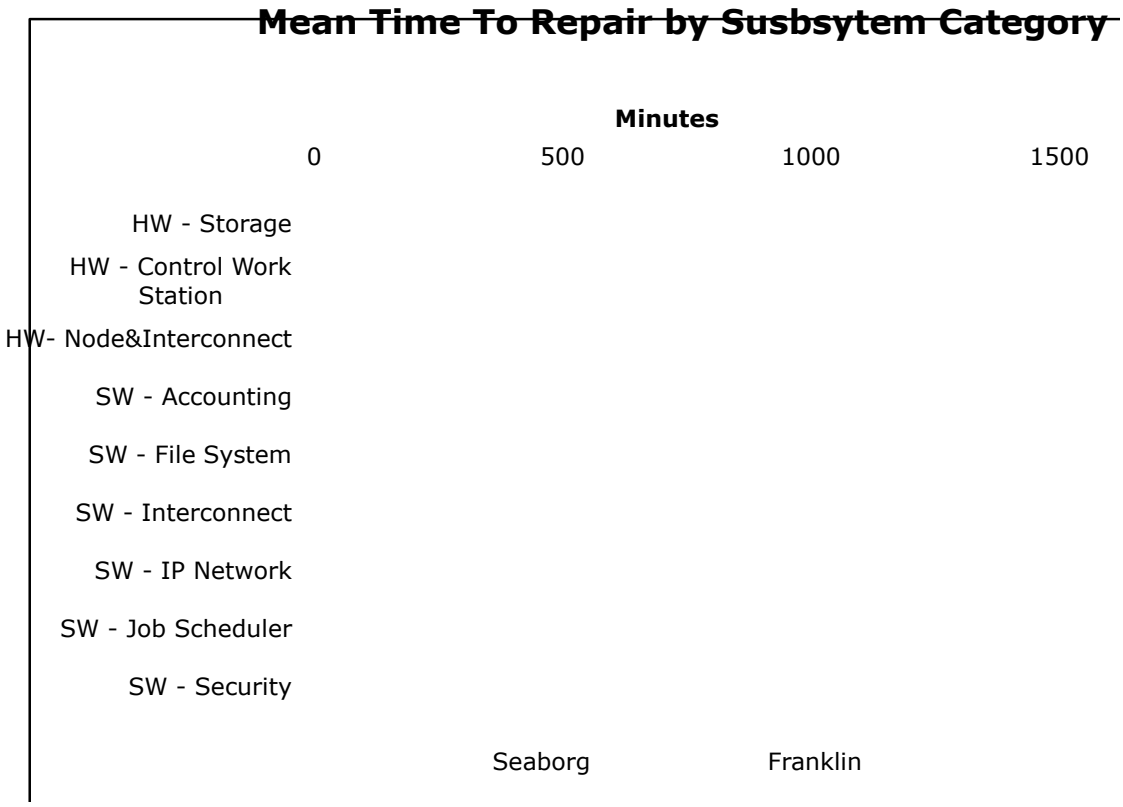


Figure 6-4: Mean Time To Repair by Subsystem Category for both systems

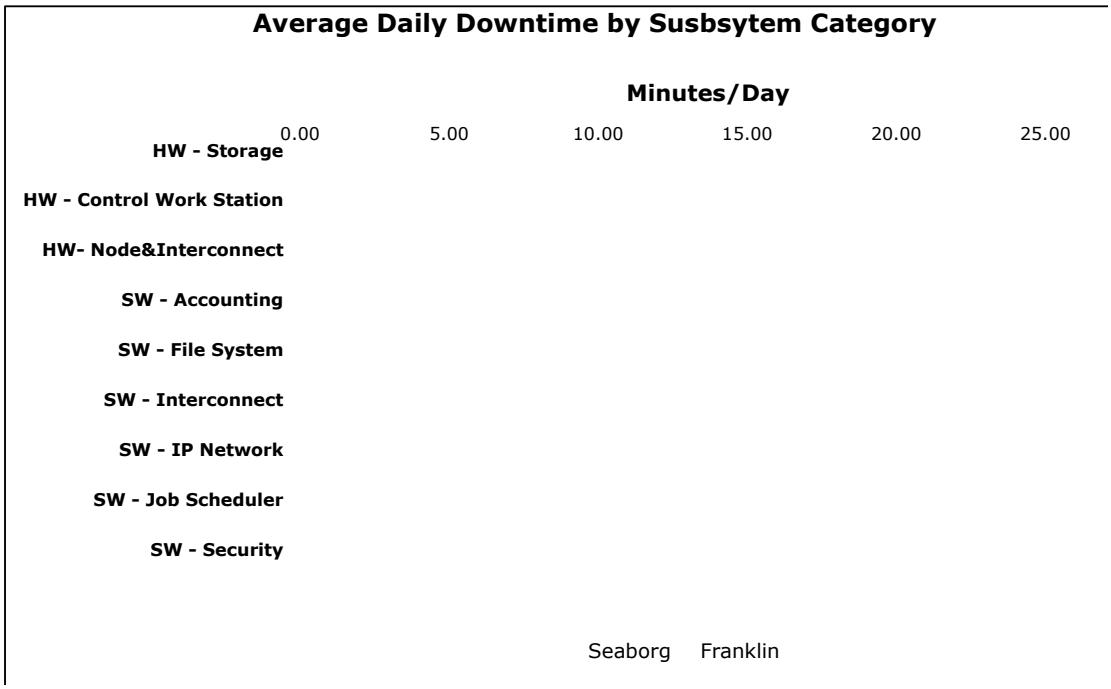


Figure 6-5: Average downtime per day for major subsystem categories.

Figure 6-4 shows the Mean Time To Repair (MTTR) by subsystem for both systems. The data is independent of the data collection period. MTTR is calculated as the total downtime divided by the number of incidents. Overall, the outages on Franklin are significantly shorter because it is possible to bring Franklin up in less than 25% of the time it takes to boot Seaborg. The large MTTR for Seaborg Security is the result of two events, one of which required a complete system build that took more than a week, in part due to the complexity of Seaborg's rebuild process.

6.3.2 Seaborg Failure Analysis

A detailed analysis of the NERSC Seaborg system showed the causes of relatively large outage times caused by software. Figure 6-6 shows outages and highlights tickets with a large downtime for Seaborg. The security incident in 2006 and the operating system upgrade in 2004, account for significant portions of the annual outages for those years, one planned the other not.

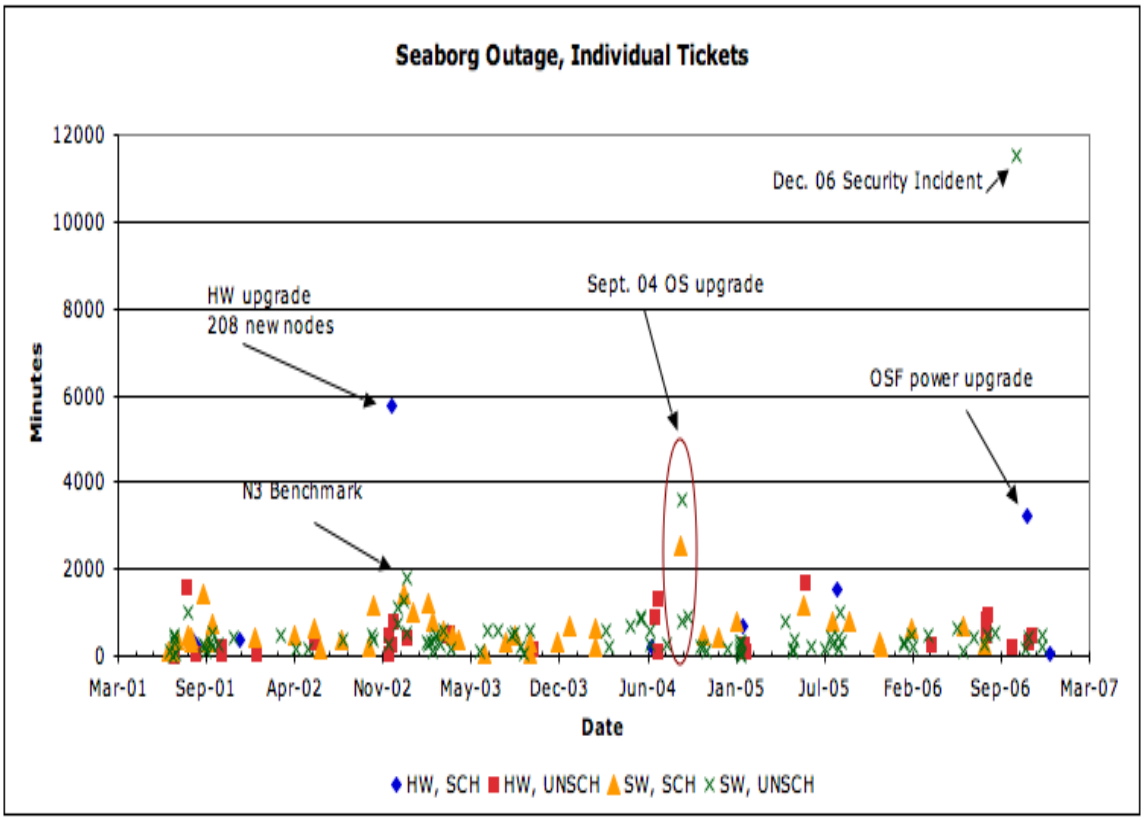


Figure 6-6: Classification of individual tickets for the NERSC Seaborg System

As is indicated in Figure 6-6, there are a number of components that can contribute to system downtime. Figure 6-3 shows the outages categorized by components, software, and hardware. The file system makes a large

contribution to the system outages. There are a number of components failures that manifest themselves as file system failure, including problems with any of the Virtual Shared Disks (VSD) connected to 20 I/O nodes which make the file system unavailable. The tracking tickets did not always reflect the exact cause, but rather the mostly likely cause based on the judgment of the system managers. The same ambiguity existed with “switch” related outages. Accounting, benchmarking, and dedicated tests also made the system unavailable to users.

6.3.3 Seaborg Node Disk Failures

One result of analyzing component failure rates is to determine the relationship between component reliability and system reliability. Assessment of individual trouble tickets showed insufficient information to indicate the action taken for the failed components in the NERSC failure data base. For example, in some cases the information available did not specify whether the vendor or system managers replaced the failed component or returned it to service.

In practice, corrective action for a failed disk varies markedly between vendors, and even system engineers within the same vendor. Thus, vendor A’s process may normally result in simply reseating or power-cycling the disk to clear the fault and returning it to service, where a failed disk on vendor B’s system might normally result in a permanent replacement. Such processes change over time. If experience shows a trend of correctable single bit errors

eventually results in a hard error, the repair procedure may evolve to replacement upon single bit errors as a proactive measure. Given these differences, this study only selected trouble tickets that indicated drive replacement in order to compare similar failure statistics for tape and disk.

The annual replacement rate of disk drives for Seaborg shows a rate of about 1.5% per year. This is lower than the observed values of 2-6% in other studies of disk failures (Pinheir, Weber and Barroso 2007) (Schroeder and Gibson 2007) albeit the population of disks is smaller than the other studies and from a single vendor (recognizing the storage subcomponents may come from different sources). It is also possible that not all disk replacement instances have been captured in this analysis since hardware technicians may have performed proactive replacement based on tracking drive correctable errors prior to being reported or detected as a drive failure. This type of maintenance is not accounted for in this study.

Figure 6-9 shows the total number and percentage of disk drives replaced in Seaborg. The total number of disks in the system over time, shown in Figure 6-7, was used to normalize the data. The majority of disks are Serial Storage Architecture (SSA) disks and only 120 Fibre Channel disks were added in mid 2004. Figure 6-8 shows the monthly disks replacement by disk type. The large number of failures in 2003 can be explained as a combination of drive age and “infant mortality” of the major increase in new drives added in 2003. This also explains the low replacement rate in 2004. Observe the

steady rate increases as drives get “older” leading to an increase in replacement rate in 2006. This trend of device failure correlated with age is also seen in the Schroeder-Gibson work noted Section 6.5, which concludes that the common wisdom that infant mortality is high and then disks stabilizes for a long period (the “bathtub” profile) is a misconception.

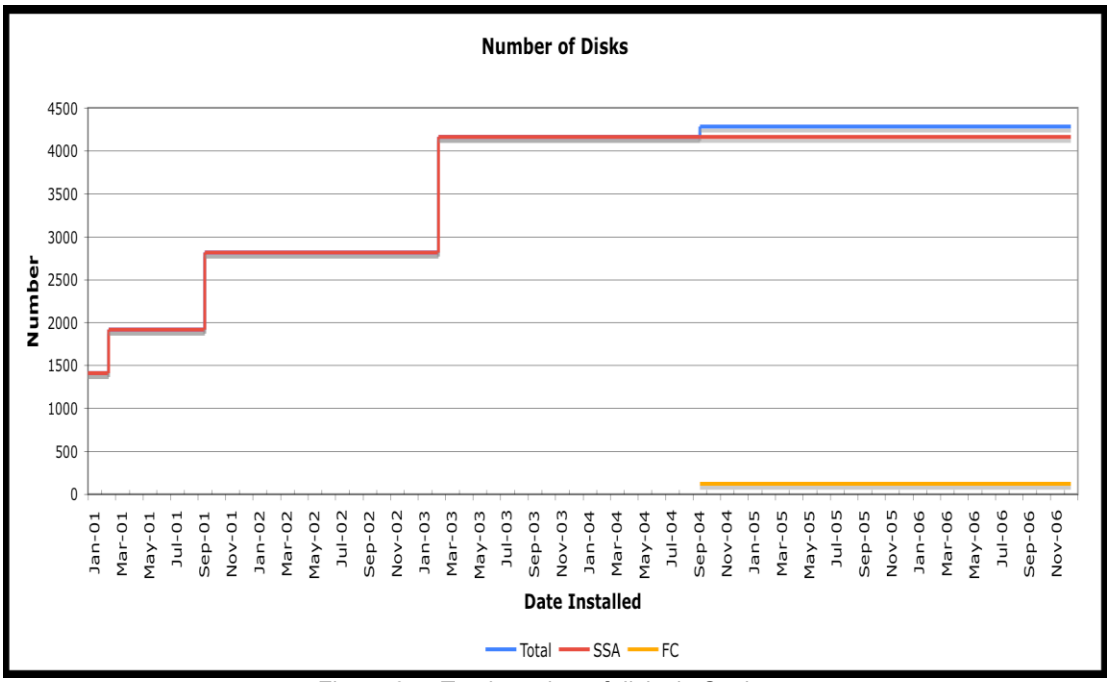


Figure 6-7: Total number of disks in Seaborg

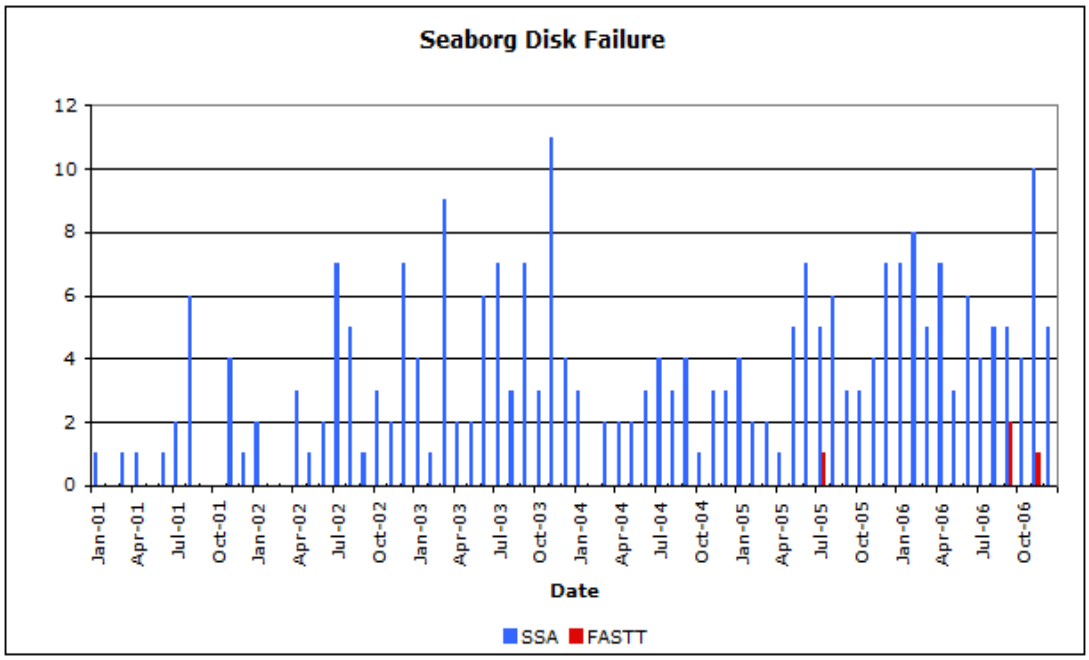


Figure 6-8: Seaborg Disk Replacements by Disk Type

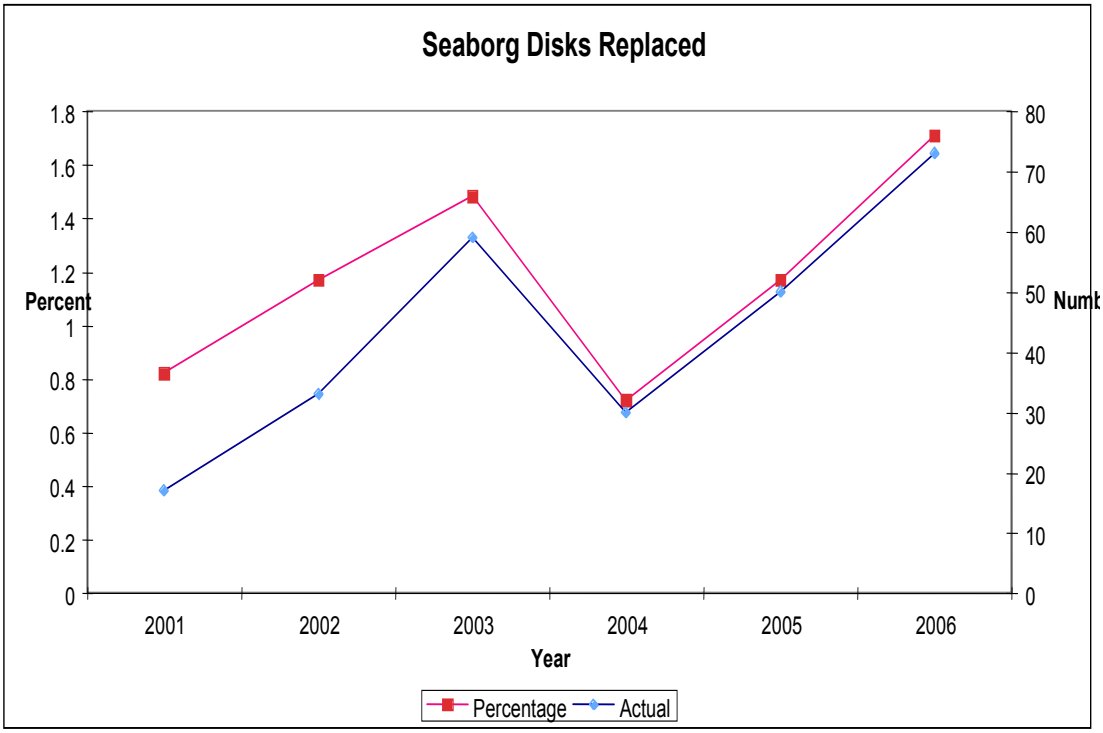


Figure 6-9: Disk Replacement Trends for Seaborg shows the increase of disk failures due to age.

6.4 Job Completion Success on NERSC-5

So far, the failure discussions have focused on system wide outages and replacement of failed components. Another important category of failure is those that disrupt part of the workload but leave the overall system operating, albeit in some degraded manner. Some failures are transparent to applications, but many are not. In particular, parallel jobs are very susceptible to single component failure unless the system masks it from the application. For example, if there is a known rate of failure of individual CPUs, then a parallel application running at twice the CPU concurrency has twice the likelihood of failing due to a CPU fault. With applications using thousands of CPUs – and looking at using millions, the fault rate of a single component becomes very obvious[*]. Hence, job completion rates are part of on-going metrics for NERSC-5.

As noted in Section 6.2 above, some system wide outages under CVN became job failures under CLE. Job success rates for CLE were collected and analyzed. The basic logistics of evaluating job completion was more complicated than first envisioned. Initially, CLE provided inconsistent and incomplete error messages due in part to faulty error message propagation across layers of the software stack. Many inconsistencies were resolved within the evaluation period so more accurate error messages were

[*] A recent example is a job at NERSC that used 9,000 nodes and was intended to run in 12 hours. A single ill behaving node, undetected at launch time, caused the job to hang and not progress. This meant no work at all was done on the system for that time. The ill behaving node had a hard failure several days later.

produced. The completion status of jobs is traced from logs and system process logs. Table 6-1 below shows the job success rate between Sept 18, 2007 and April 11, 2008 – more than ½ a year from the early acceptance testing of CLE through production usage. Unlike other job completion metrics that assess how often the exact same application/problem set completes successfully, this metric deals with real jobs representing 3,000 users doing actual science and over 500 different application codes running on the production system. During this time, 178,133 significant computational jobs ran on the system.

Failure Error Category	Number of Jobs	Percent of Jobs
SUCCESS - Job clearly succeeds.	117,884	66.2%
WALLTIME - Job ran to the wall clock time limit. A number of users let the job run out of time intentionally. However, there are cases where a node assigned to the job is in ill health but has not yet been detected, causing the job to go very slowly or hang with no progress.	12,614	7.1%
WIDTH - A mismatch between what the job requested and what the aprun command uses -– normally a user error.	0	0.0%
NODEFAIL – A node assigned to the job failed or crashed – possibly hardware.	192	0.1%
UNEX - This error indicates MPI buffers need to be increased.	75	>0.05%
ENOENT – A requested executable file does not exist.	1,148	0.6%
LIBSMA - An error within the SHMEM communication library.	70	>0.05%
SIGTERM - Job received a Terminate Signal (Kill -9). This could have been from the user or the system.	58	>0.05%
NOAPRUN - The batch job did not appear to execute an aprun. This is usually due to a batch scripting error.	6,516	3.7%
NOTRACE – For some unidentified reason, process accounting data could not be traced to identify the aprun associated with this job. The job did execute an aprun but the parent process id was 1 so it could not be properly matched. The usual cause is that a job was killed and the last process to exit was aprun so its ppid was 1.	11,389	6.4%
QUOTA - Job exceeded a File System quota.	2,865	1.6%
ATOMIC – The job failed due to a software problem when using parts of the SHMEM library (the problem has since been fixed).	4	>0.05%
UNKNOWN - The status of the job completion was non-determinate. What is known is the aprun command had a non-zero exit code. This may be due to a system problem but more likely due to some user action that prevents recording the exit status in system logs, e.g. an application trapping a signal or redirecting I/O.	25,318	14.2%
Total	178,133	

Table 6-1: Job Failure Error Categories and Data from Sept 2007 to April 2008.

6.4.1 Manual Analysis of Job Failure Data

A subset of the failing jobs – several hundred – was manually analyzed in detail. Users who submitted the job were contacted to determine if the failure was intentional, in the application, or system generated.

This investigation led to several conclusions:

- Root cause job failure analysis is very time consuming for support staff and users, so it is not tractable to do a full analysis of every job failure. Automation is required and with that there is some potential for mis-categorization.

- NERSC has sophisticated users who can determine the cause of errors in their runs and proactively report suspect job failures that are not due to their error with reasonably high degrees of accuracy. Most batches of system level errors correspond to increased user problem tickets.

- A significant number (~20-30%) of errors were due to user mistakes or code problems. However, each error category had job failures due to system issues. Categorizing a type of error due only to user errors or only to system errors with complete accuracy is not possible for most categories given the level of information being reported by the kernel and related job management processes.

    - For example, while many WALLTIME errors were under user control; “hung” nodes or other diagnosed system errors also caused jobs to start, make no progress for their entire time slot, and exit, giving the same error message. Hence, not all WALLTIME exceeded messages were user problems.

    - Another example of a category that is mostly user, but sometimes system issues is over running file quota. This is

typically is considered a user error, but the system generated 49 such errors since January 2008 despite having the quota function entirely turned off while awaiting bug fixes. These are a system-generated error.

- WALLTIME, WIDTH, SIGTERM, NOAPRUN, are now considered likely user generated unless there is a pattern detected when many jobs generate the same error. QUOTA will be in the user category once it is functional.

- NODEFAIL and ATOMIC is clearly system issues.

- NOTRACE is associated with jobs that terminate during a system crash. The NOTRACE label is because the jobs cannot write out a record.

- UNKNOWN represents a significant number of failures and is troubling since it means exit status could not be automatically determined. It should be possible for the system to reliably record all process exit codes for post mortem analysis. It remains a goal to drastically reduce the number of unknown conditions for job exits.

Single Day Job Success Data			
From: 10/16/08 00:03:27 to: 10/16/08 23:57:30			
Job Exit Status	Job Count	Percent of Job for Day	Estimated Fault Cause
APINFO_SUCCESS	580	63.7	N/A
APINFO_TORQUEWALLTIME	79	8.7	User
APINFO_APRUNWIDTH	0	0.0	User
APINFO_NODEFAIL	1	0.1	System
APINFO_MPICHUNEXBUFFERSIZE	0	0.0	User
APINFO_ENOENT	6	0.7	User
APINFO_LIBSMA	0	0.0	User
APINFO_SIGTERM	0	0.0	User
APINFO_NOAPRUN	28	3.1	User
APINFO_UNKNOWN	69	7.6	Unknown
APINFO_NOTRACE	36	4.0	Unknown
APINFO_SHMEMATOMIC	0	0.0	System
APINFO_DISKQUOTA	0	0.0	System
APINFO_SIGSEGV	1	0.1	User
APINFO_CLAIM	7	0.8	User
APINFO_MPIABORT	40	4.4	User
APINFO_NIDTERM	65	7.1	System
APINFO_ROMIO	0	0.0	User
APINFO_MPIIO	0	0.0	User
APINFO_BOGUS	0	0.0	Unknown
Total	912		

Table 6-2 Job Success and Failure indicators for a single day. (Data courtesy of Mr. Nicholas Cardo, NERSC)

Table 6-2 shows a single day's job success and failure rate for October 16, 2008 on the NERSC *Franklin* system. While each day fluctuates, this day is typical of others during this time period. The general trend is about 60-65% of the job exit successfully with a code of 0. About 5-7% have suspected system causes for termination and between 10-20% have causes that cannot be determined automatically. User caused job termination is 10-20% of the jobs.

6.4.2 Observations About Job Completion Metrics

Job completion metrics were unexpectedly difficult to accurately assess in the automated manner that is necessary on large systems. Work continues to more accurately report and diagnose errors. Despite the difficulties, tracking job exit codes is valuable in diagnosing and correcting many system faults. At the moment, looking for patterns such as large increases in the percent of a particular category then merits manual investigation.

6.5 Reactive Assessment of Reliability

In many evaluations, reliability assessment is reactive in the sense evaluators have a system that is placed in service and then they measure such attributes as system availability, node availability, Mean Time To Interrupt* (MTTI), and Mean Time To Restoration of Service (MTTR). Mean-Time-Between-Failure and Mean-Time-Between-Interrupt are similar reliability measures. Furthermore, such measures may not be from the end user point of view, but rather from the system provider point of view.

After the system is in place, particularly in production use, it is common to have regressive metrics (penalties) for the service providers if expected reliability is not being met. Unfortunately, even the current estimates of component reliability are inaccurate. (Schroeder and Gibson 2007)show hardware disk vendors supplied estimates of disk failure rates differ as much

* Other sites use Mean-Time-To-Failure (MTTF). Because of the many redundancy features in large systems, MTTF is appropriate to analyze component failures, but not system quality of service since a system can often continue to operate with some level of failure.

as 15x from the reality seen in large scale facilities, with a difference of 5x being common.

(Gonzalez, et al. 2007) categorizes failures into three areas – transient, intermittent and permanent. Transient errors appear for a short time, and then disappear. Intermittent errors appear and disappear. Permanent errors persist. An issue is how much effort needs to be applied to each type of failure. Current modern systems are tending to exhibit increasing numbers of transient and/or intermittent errors. One supporting fact for this is (Schroeder and Gibson, 2007) 43% of disk drives returned to manufacturers after a “hard” failure in the field do not exhibit the failure when analyzed in the vendor’s facility. The transient and intermittent errors make traditional reliability assessment methods (Nagaraja, et al. March 2003) less effective.

6.6 Proactive Assessment of Reliability

The HPC community struggles to find a proactive method of assessing reliability. Most evaluations do not attempt proactive metrics. Methods such as fault injection require understanding fault profiles as well as representative systems available for running the test. But without clear data, such profiles may be based on incorrect assumptions. Further, with the multi-million dollar expense of HPC systems, it is seldom feasible to do fault injection studies for full systems, so statistical and pattern recognition methods are more likely to result in improvements.

6.6.1 Assessing A System Provider's Response To Errors

One method to evaluate reliability may be not trying to assess all faults and reliability directly, but rather to assess a system provider's ability to understand and respond to hardware and software errors. This approach was taken in the NERSC-5 procurement (NERSC-5 2004), which asked the vendors to "*Provide information concerning the number of defects filed at each severity level and average time to problem resolution for all major software and hardware components.*"

The responses to this question varied widely. Some system providers demonstrated they had detailed data for problem analysis, including sub-component classification of errors for both hardware and software. Other respondents provided general system wide information. One vendor refused to provide any data since they felt it could be used to "misrepresent individual products." Nonetheless, proactive assessment of reliability will continue since being able to understand the statistics of system failures is a prerequisite to effectively addressing them.

6.6.2 Size Of System Provider's Testing Environment

Another proactive measure being used on the NERSC-6 evaluation is the size and completeness of the system provider's testing environment. This includes the relative size of the test environment to the system being offered, the robustness of the testing methods, etc.

6.7 Observations About The Importance Of Reliability Data Collection

The collection and analysis of reliability data is becoming more difficult due to the volumes of data and the complexities of large scale systems, while at the same time, more valuable. It is even more critical because large systems are composed of significant layered software, often open source software, and there is much less integrated testing across identical hardware and software configurations. Even when the data is supplied by individual vendors, the software layers can vary considerably between systems. Hence, without the ability to have a large number of identical system images, the ability to mine data is important to improving system reliability.

6.8 Related Work

Work from Rutgers (Nagaraja, et al. March 2003) and others groups discuss methods to assess “Performability”. Many of the papers concern assessment and modeling availability of small scale systems. Performability is defined as a system’s performance multiplied by its availability, which makes it similar to Potency. The Rutgers work assesses systems at a relatively high level, with the assumption that many low level faults are masked or handled by hardware and/or software before they impact applications. So, they look at modes of failure at the node and commodity network levels. They developed a model to determine average availability and throughput of systems which assesses throughput for each type of component failure. Different failure modes are modeled and tools are used to

induce the expected failures in a running test system which then is supposed to show changes in performance.

The Rutgers method has potential application to assess the expected reliability of large systems. While Rutgers assesses small web server clusters, conceptually it can be carried to HPC systems. However, there are limitations in the Rutgers work that require extensions and may even make it intractable for application to current and future HPC systems. The first limitation is the assumption that all faults are independent. In a farm of web servers, this may be so, since one server going down is not likely to cause another to crash. HPC systems have much tighter integration and there are many “system wide” failure examples from which one component generates a cascade of problems in addition to single component failures. It is also unfortunately relatively common to have the entire HPC system fail without a cascade from a single component, another consideration that is not the main thrust of the Rutgers work.

Another assumption the Rutgers group uses for simplification is that MTTR is much smaller than MTTF. This simplifies their analysis and modeling but it may not be the case in HPC systems, where a failure may take the system out for hours, days or even longer[*].

[*] Current large IBM systems take 2-4 hours to restart. A file system check of 20 TB file systems may take upon to 12 hours or more. A modest size commodity cluster takes about 30 minutes to boot.

Other works in the literature involve ‘black box” testing and/or fault injection into systems that is not tractable for testing methods in HPC since many failure modes are seen only at scale. The HPC industry does not have the resources to devote large systems to such systematic black box testing. Indeed, many vendors now only have small systems in house for all their testing needs.

One important related, but not direct, HPC effort, dealing with reliability is the Recovery Oriented Computing (ROCs) (ROCS n.d.) and Reliable Distributed System Laboratory (RADLab) (RAD Lab n.d.) efforts led by Dr. David Patterson and Dr. Armando Fox. These projects focus on web service farms (Fox, Kician and Patterson 2004) and other loosely distributed systems but are related to HPC in the scale of components and complexity of software. The methods being explored in these efforts are areas such as managing and analysis of high volume log streams (Xu, et al. 2005). Another promising area of research is using statistical learning theory to monitor quality of service and provide early detection of potential failures (Xu, Bodik and Patterson November 2004). The HPC community would be well served to explore these efforts and others that are similar in nature.

Another related area is an increasing body of work coming for reliability, serviceability, and performability studies in commercial web serving facilities (Pinheiro, Weber and Barroso February 2007). This publication of disk failure analysis in large web systems generated other articles on the impact of

storage. There is more in common between the HPC community and the web services community that merits exploration.

Fortunately, there are some groups focused on HPC reliability issues directly. The Petascale Data Storage Institute (PDSI 2007) expanded its reliability studies to all aspects of HPC systems (Schroeder and Gibson 2007) – not just storage. Papers and failure data from HPC systems is available for the community in conjunction with the Computer Failure Data Repository (CFDR) (CFDR 2006). At the CDFR project, seven failure data sets for HPC systems and clusters are available as well as data from several web providers. These data sets, several used in this work, became openly available since 2006, and papers are starting to emerge based on the data.

Other HPC specific efforts that involve not only reliability, but other aspects as well, are also active. Two of note, devoted to Petascale Systems, are the 2007 Petascale System Integration Workshop (PSIW 2007)and Report (Kramer, et al. 2007) and the 2008 Risk Management Techniques and Practices Workshop (RTMAP 2008) that will have a report of forthcoming. These will probably be followed with other related efforts.

6.9 Chapter Conclusion

Reliability is a key aspect of system productivity that is typically studied independently. This chapter discusses the issues of reliability for HPC systems with the major points being:

- Software failures are the dominant cause of system wide outages on HPC systems.

- Traditional analysis of reliability is not providing sufficient insight into the probability a large-scale system will be able to provide reliable service.

- In a general production environment, application failure rates due to system problems is difficult to accurately assess, but can be very worthwhile, even if not precise.

- Having demonstratable processes to accumulate and assess failure information is a potential proactive indicator of a system's probability of providing reliable service. Few vendors have the demonstrated this ability, particularly for software.

Chapter 7: Consistency of Performance

7.1 Chapter Summary

The design and evaluation of high performance computers concentrates on increasing computational performance for applications. Performance is often measured on an optimally configured, dedicated system to show the best case in performance, often in the space of a few hours to a day or two. In real environments, resources are seldom dedicated to a single task and systems run multiple tasks that may influence each other.

Furthermore, many factors influence large scale systems that may significantly impact achieving performance in a consistent manner, including interconnects, topology, congestion aware messaging, assignment of memory and software jitter. Hence runtimes vary, sometimes to an unreasonable large extent.

But what level of consistency is reasonable to expect for HPC? In this chapter, comparisons are made across several architectures, interconnects and operating systems for large distributed memory systems in a systematic manner. It then analyzes the causes for inconsistency and discusses what can be done to decrease the variation without impacting performance. Finally discussed are issues of on-going assessment of consistency through the system's life cycle.

7.1.1 Motivation For Consistency

Inconsistency of parallel application performance has broad implications for how much useful work can be produced by a particular HPC system. Factors leading to changes in performance occur over multiple time scales and originate both from within applications and from external sources. As a result, variability in runtime performance is strongly tied to the hardware and software architecture. This work examines performance consistency in parallel applications on time scales of years to microseconds, with a focus on understanding some of the causes of performance inconsistency in multi-user production environments. Where possible, specific causes for the inconsistency observed are identified.

Benchmarking and workload characterization may be taken either on dedicated hardware or in the context of non-dedicated, day to day use. It is the more complicated production context that is arguably more important in setting user expectations about how long a scientific calculation will take to complete rather than focusing on improving performance by a modest percentage in dedicated testing. As things scale more, due to multi-core technologies and increasingly parallel systems, the likelihood of inconsistency increases unless proactive steps are taken.

7.1.2 Factors That Influence Consistency

One goal of the present chapter is to identify some of the factors which influence the probability a system will be consistent. In order to do so, it is important to be able to detect, measure, and address the causes of performance variability. Keeping identification and quantification goals in mind, this chapter first concentrates on how the shape of the distribution of runtimes for a task or application is influenced by a variety of factors and events occurring on the system.

Understanding the parallel scaling factors leading to performance inconsistency is a chief concern of those who use and maintain large scale parallel computers. Large scale HPC resources are built from thousands of smaller systems. Since the majority of testing and performance analysis is done on test systems much smaller than production machines, it is common to encounter variability induced performance loss that goes unseen on smaller systems. As seen in the following sections, the performance impact of inconsistency can be quite large, becoming the dominant impediment to parallel scaling in some cases.

7.2 The Impact of Inconsistent Performance

Application performance for computer systems, including studies of parallel system hardware and architectures, is well studied. Every architectural feature is assessed with application and specialized benchmarks – be it on real or simulated systems – so the impact of the functions can be

evaluated for its performance impacts. To a much lesser degree, system software is evaluated for performance of both the system software itself and the applications that use it.

7.2.1 Users Are Impacted By Inconsistency

The variability of performance is as important as availability and mean time between failures to users to be able to accomplish their goals. For example, the user's productivity is impacted at least as much when performance varies by a factor of two, as when a system's availability is only ½ of the expected time. In both cases, the amount of work done is only half of what is expected of the system.

7.2.2 Negative Impacts Of Inconsistency

Multiple sources show inconsistency in runtimes leads to many negative impacts [(Figueira and Berman 1966), (Worley and Levesque 2004), (Zhang, et al. 2001)] all of which make a HPC system have less value. The first impact is less overall work done by the system. Runtime inconsistency is inherently bad for performance since variations in runtime proceed upward from some best case runtime, i.e., variation is seldom toward better than optimal performance. The longer a task takes, the more time it takes to get usable results for analysis. Since some applications have a strict order of processing steps (i.e. in climate studies, year 1 has to be simulated before year 2 can

start), they cannot directly overcome this slowdown via, say, increased parallelism.

Inconsistency decreases the efficiency of HPC parallel computers since cycles are lost to both job failure and complex job scheduling to mitigate the lack of consistency [(Srinivasan, et al. 2002), (Lee, et al. 2004)]. Jobs fail through incorrect estimation of the batch queue requirements. System scheduling becomes less effective because users must be overly conservative in requesting batch time. Most scheduling software relies on user-provided run estimates, or times assigned by default values, to schedule work effectively.

When a cautious user over estimates runtime, the job scheduler operates on poor information and results in inefficient scheduling selections on systems. On the other hand, if a job request does not adequately plan for inconsistent runtimes, there can be a substantial probability the job overruns the requested batch queue time and suffers a loss of productive time elapsed since the last checkpoint. In the case of code which does not perform application check pointing, the entire run may be lost*. These all contribute to the loss of user productivity and decreased system impact.

7.3 Coefficient of Variation

A useful metric for understanding consistency is the Coefficient of Variation (CoV), defined by the standard deviation of a sample divided by the

* Many third party software packages do not provide check pointing.

arithmetic mean. Specifically, for a given number, V, of application runtime results, t_v, on a given system, then the Coefficient of Variation is defined as:

$$\bar{t} = \frac{1}{V}\sum_{v=1}^{V}\left(t_v\right)$$

$$CoV = \frac{\sqrt{\frac{1}{V}\sum_{v=1}^{V}\left(t_v - \bar{t}\right)^2}}{\bar{t}}$$

Equation 7-1: The Coefficient of Variation is the standard deviation divided by the mean of a series of observations.

The CoV has shown to be useful in a number of situations in diagnosing consistency issues on real systems [(Kramer and Ryan 2003), (Skinner and Kramer October 6-8, 2005), (Kramer and Ryan May 2003)].Other measures may also be useful at times, such as the range of minimum/maximum values, but CoV provides a measure that can be assessed over time and is independent of things like improved application performance due a changed complier.

7.4 Consistency of Two Light Weight Operating Systems on the Cray XT-4

Before fully exploring consistency in a number of contexts, it is interesting to look at consistency for the NERSC-5 system as it applies to two different Lightweight Operating Systems – Cray Virtual Nodes (CVN) and Cray Linux Environment (CLE).

The average CoV across the SSP-4 applications was 0.4% for CLE and 0.35% for CVN – remarkably close considering CLE was derived from a Linux operating system. The CoV was calculated for each SSP-4 application by doing a modest number (~20) multiple runs of the same application and problem set, and then, these individual CoV's were averaged. The LWOS CoV is significantly less than that found on other full OS configurations discussed below. It is also much lower than the other light weight kernel implementation discussed in Section 5.2, leading one to conclude that both LWOS implementations were carefully designed to limit variation.

However, other tests show significant decreases in consistency with CLE, particularly shorter running tests such as the NAS Parallel Benchmarks. Stream, particularly the version of the streams test that use less than 50% of available memory, showed increased variability as well as lower performance. If the ratio of CLE CoV to CVN CoV for all tests – from single core to the full configuration test - is averaged, CLE has a CoV six times that of CVN. This is opposed to about a 14% increase for the larger scale SSP applications. It is important to note the CLE CoV is still more than a factor of five lower than that observed on other systems that run full blown Linux or Unix based operating systems on different hardware.

Both CLE and CVN provide very consistent timing for applications. Under CLE, the consistency actually seems to improve with scale. Both CVN and

CLE provide an improvement in consistency over systems that use full operating systems on compute nodes.

7.5 Inconsistency Exists in Application Performance

Performance inconsistency is caused by many factors. On multi-user systems with multiple jobs running within a shared memory processor, frequent causes of inconsistency are memory contention, over scheduling the system with work, and priority of other users.

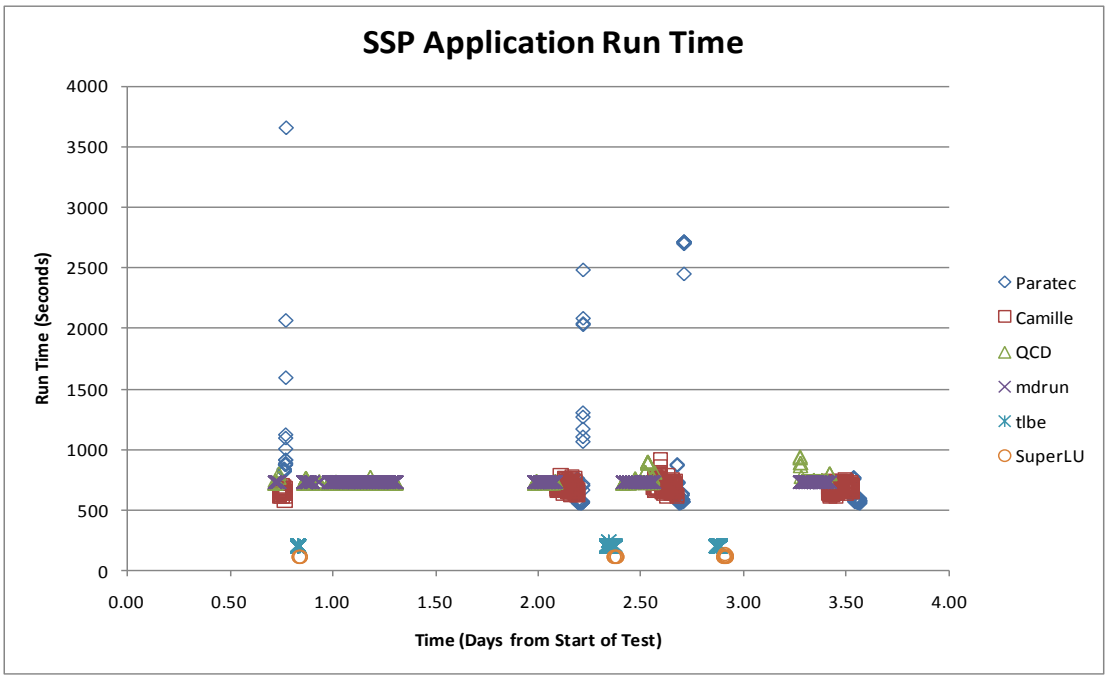


Figure 7-1: This chart shows inconsistency in performance for 6 full applications running on the NERSC IBM SP Seaborg system. These codes were part of the SSP-2 suite used for system acceptance with 256 way concurrencies.

However, on large-scale distributed memory systems, it is rare the compute-intensive parallel applications share SMP nodes. Figure 7-1 shows the run-time inconsistency present on the NERSC IBM SP “Seaborg” system

when it was first installed. The codes, run with a concurrency of 256-way, were run multiple times over a four day observational period with essentially nothing else on the system. The runtime inconsistency shows that large-scale parallel systems exhibit significant inconsistency unless carefully designed and configured. Previous experience had shown that a number of software factors could cause inconsistency, including slightly different system software installation on each of the nodes, system management event timing (daemons running at particular times) and application performance tuning. These issues were all mitigated on the system during installation and before the testing period shown. However, configuration problems, bugs, and architectural issues remained that make the system inconsistent.

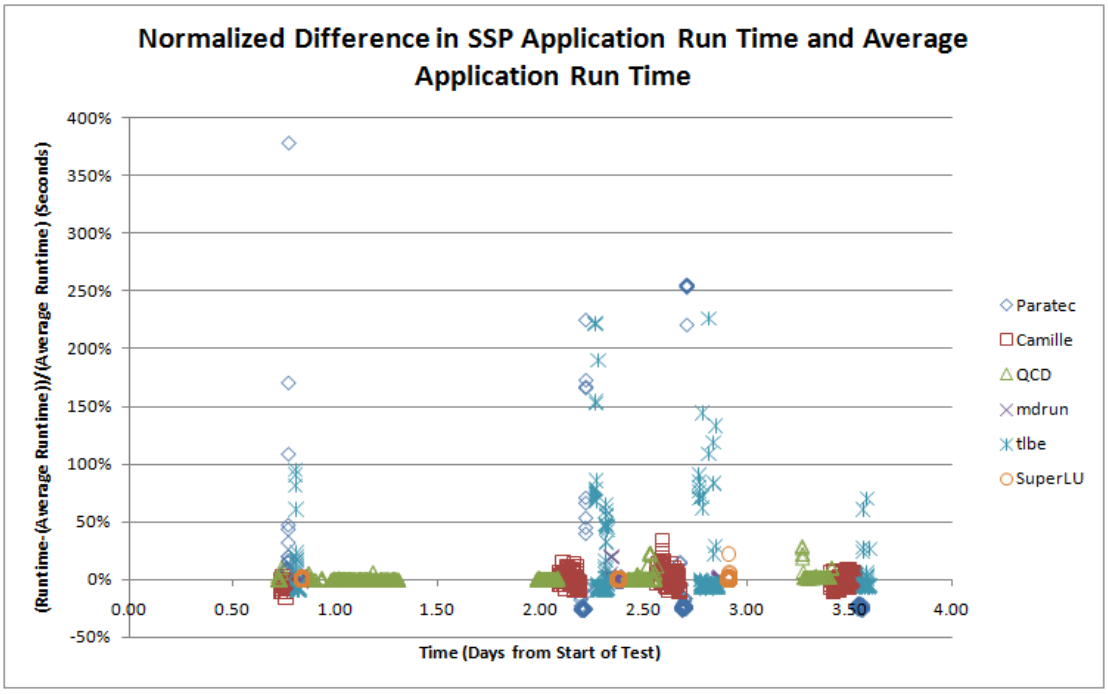


Figure 7-2: this is the same data as in Figure 7-1, showing the difference in run time of applications compared to the average run time of the applications, normalized by the average run time. All applications show some inconsistency, and several show significant inconsistency.

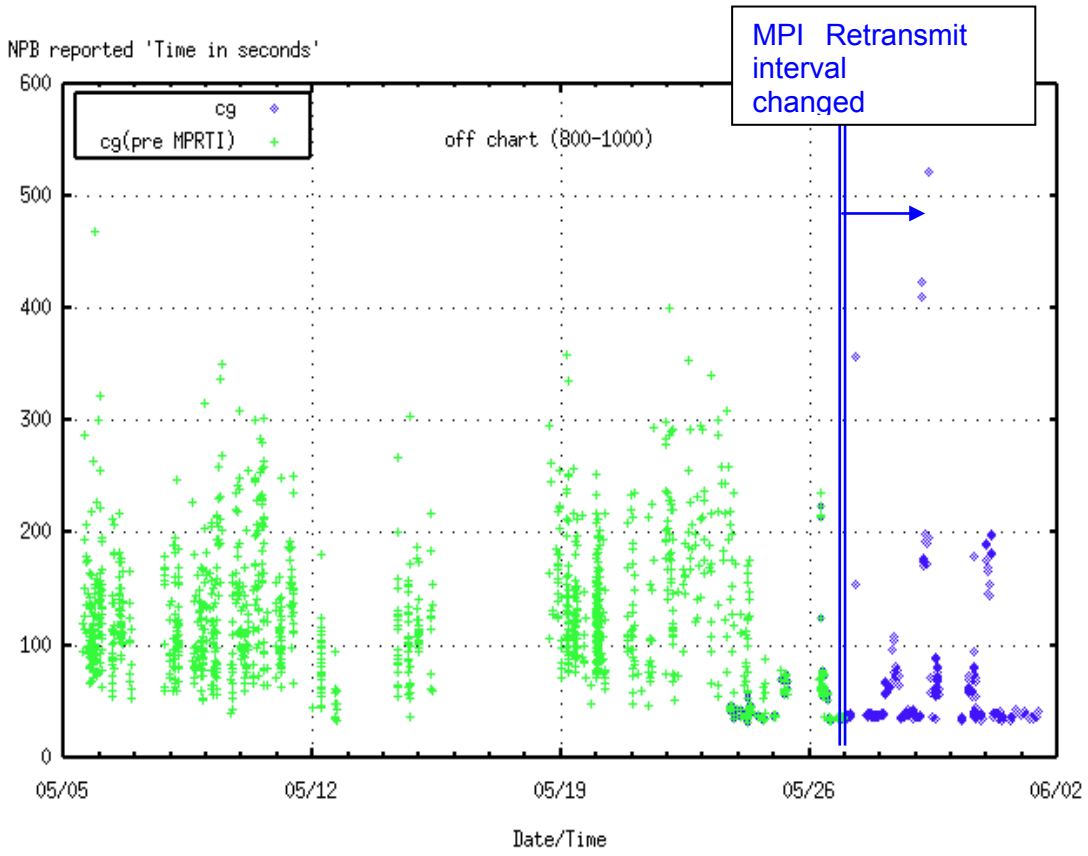


Figure 7-3: Shows the inconsistency in performance of the CG benchmark with 256-way concurrency before and after adjustments were made to the MP_RETRANSMIT_INTERVAL parameter. The interval controls how long an application waits before retransmitting MPI messages.

Figure 7-3 is another example of varying runtimes for a single code before and after tuning the High Performance Switch (HPS) switch configuration. The MPI retransmit interval tells an application how long to wait before retransmission of a message. The value was initially configured to accommodate a switch topology called “double-single.” In order to address other performance and inconsistency issues shown in Figure 7-3, the switch was adjusted in the upper levels so each upper level switch node had less traffic per link. However, this adjustment resulted in having to recalibrate the timing intervals in the switch to improve consistency.

Despite the challenges, it is possible to make very large systems operate in a consistent manner. Table 7-1 and Figure 7-3 and Figure 7-4 show the results of running the NAS Parallel Benchmarks on the same system seven months after the initial use period show in Figures 7-1 and 7-2. It shows that on a heavily used (85-95% utilization) system, the benchmarks perform very consistently over multiple runs.

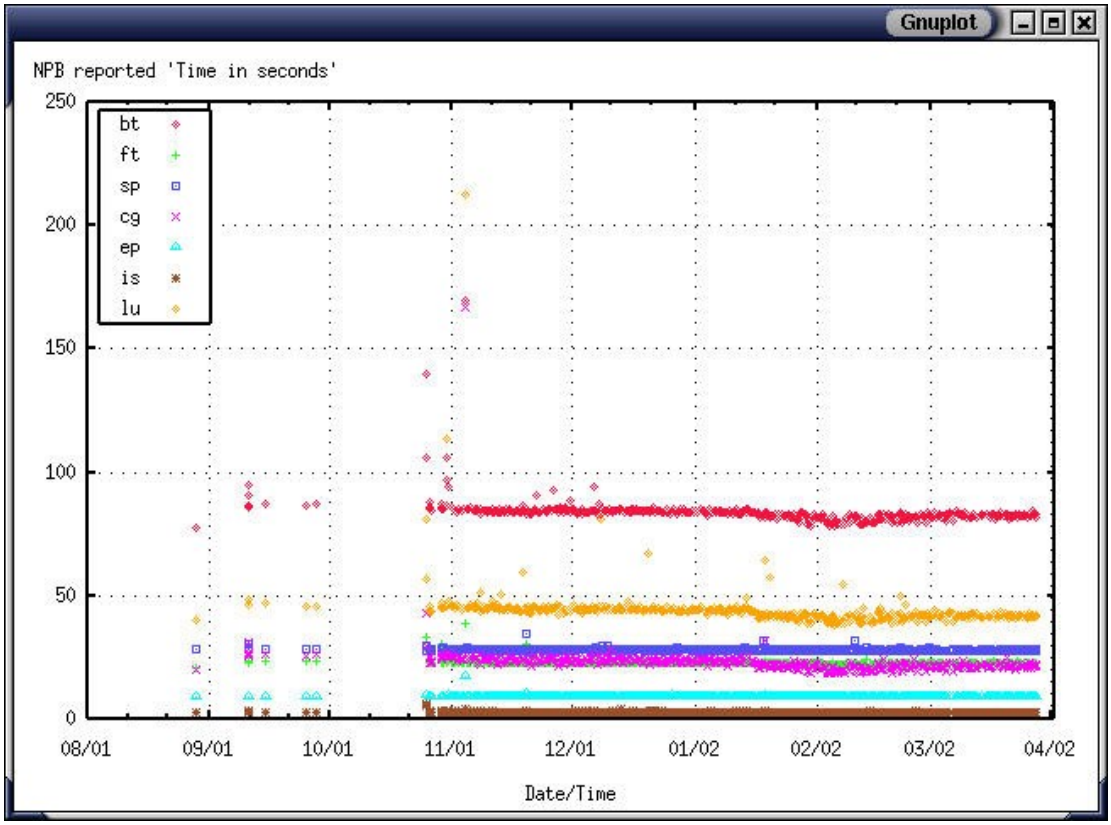


Figure 7-4: Shows seven months of runtimes for six NPB codes on the same system, all run in production, multi-user time. The graph indicates an improvement in the system consistency that was the result of multiple improvements including bug fixes and exploration of improved tuning parameters. One point of the chart is that a well configured and managed system can be very consistent.

Once inconsistency is identified, as shown in Figure 7-2, it is possible, albeit not always easy, to restore consistency by making changes to

parameters, fixing bugs and adjusting configurations so the system is well-configured and well managed.

Code	Min	Max	Mean	Std Dev	Coefficient of Variation
BT	80.80	84.13	82.42	0.64	0.78%
FT	22.17	23.57	22.42	0.22	0.99%
CG	20.22	24.59	21.39	0.70	3.25%
EP	8.57	8.74	8.61	0.04	0.48%
LU	40.70	43.14	41.75	0.56	1.38%
SP	27.31	28.45	27.73	0.21	0.77%

Table 7-1: Runtimes (in seconds) reported by the NAS Parallel Benchmarks using 256 way concurrency for the last 50 days of the period covered by Figure 7-3.

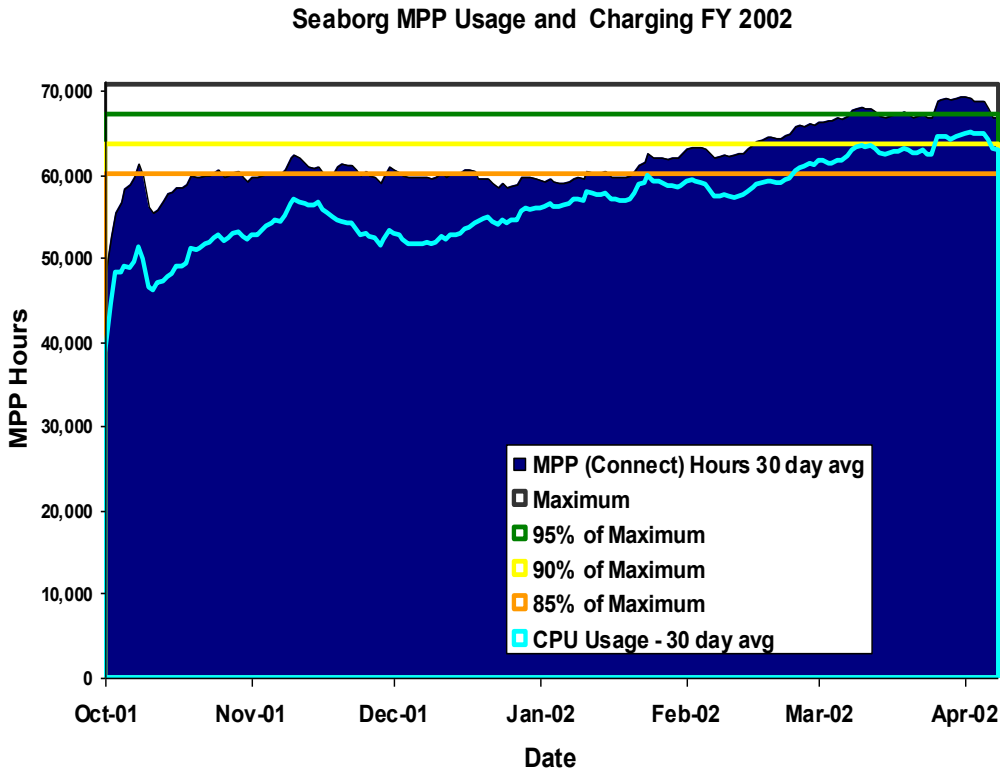


Figure 7-5: The computational load across the entire NERSC IBM SP Seaborg system, including the time period covered in Figure 7-3. The system is heavily utilized by compute-intensive applications, which received over 90% of the overall CPU cycles.

7.6 How Much Consistency Should Be Expected?

As noted above, it is possible to maximize performance consistency, including well organized system administration and management nodes dedicated to single application, eliminating bugs, adding more resources and configuration tuning. Nonetheless, questions remain about how much inconsistency is acceptable, how consistency can be achieved on specific systems, and what most influences consistency.

7.6.1 System Architecture Influences Performance Consistency

Of the benchmark suites available, an effective one for this purpose is the NAS Parallel Benchmarks (NPBs), created at NASA Ames Research Center. The NPB benchmarks have been heavily used on a wide range of architectures. They are portable and are known to give comparable results across systems. The NPBs evolved from Version 1 to Version 2 and have been used for over 10 years, so they are well understood. Finally, the benchmarks have been correlated to real scientific workloads at a number of sites.

For the sake of simplicity, three NPB benchmarks, LU (Lower Upper), FT (Fourier Transform) and EP (Embarrassingly Parallel) were chosen for use from the 2.3 version of the NPB's.

7.6.2 Architectures Evaluated

Four systems with different architectural features were used in the evaluation. The complete list of system features are listed in Appendix B but a brief summary is provided here.

Cray T3E (Scott and Thorson 1996) - The oldest system in the study consists of 696 CPUs, each capable of 900 MFlops/s of peak performance and had 256 MB of local memory. The CPUs (PEs) are connected by a network arranged in a 3-dimensional Torus with low latency (4.3 microseconds) and relatively high bandwidth (~300 MB/s bandwidth) per adaptor with static routing. The system used a UNIX like operating system that has a Chorus derived microkernel on the 644 compute nodes and UNICOS/mk on the OS and command nodes.

IBM SP [(Amos, Deshpande, et al., RS/6000 SP 375 MHz Power3 SMP High Node 2001), (Barrios, et al. December 1999)] - The largest system in the study was the 6,756 processor IBM-RS/6000-SP with 184 compute nodes containing 16 Power 3+ processors connected to each other with a high-bandwidth, switching network known as the “Colony” switch in an Omega (fat tree) topology (Wu and Fend August 1980). Each node ran a full instance of the Unix based AIX operating system runs on every node. Each node has two switch adapters. Sixty-four nodes had 32 GBytes of memory, four nodes had 64 GBytes of memory and the remaining 380 have 16 GB of memory.

Compaq SC (Hoise n.d.) - The Compaq SC system at the Pittsburgh Supercomputer Center (PSC) was composed of 750 Compaq Alphaserver ES45 nodes each containing four 1-GHz processors capable of two Flops per cycle and runs the Tru64 Unix operating system. This system had the highest memory bandwidth and the CPUs had the highest peak operations rate. A Quadrics (Beecroft, et al. 2003) interconnection network connects the nodes in a Fat Tree topology.

IBM Netfinity Commodity Cluster (IBM, Inc. June 2008). The smallest system in the study was a commodity cluster of 85 two-way SMP Pentium III nodes connected with Myrinet 2000, another fat tree. Each node runs the Linux RedHat distribution and this system did not run at full utilization.

7.6.3 Evaluation Results

The systems analyzed were four types of architectures and two types of network topologies; 3D torus and fat trees. On each system, multiple runs were executed for each of the three NPB codes – LU, FT and EP. All codes were run using the Class C problem sets with a concurrency of 128. In this case, *128 way concurrency* means using 128 MPI tasks. This was chosen because it used at least 8 nodes on the system with the largest SMP nodes and it ran long enough to minimize the effects of start-up events. Mixed mode programming - combining MPI and OpenMP - was not evaluated. The jobs were run in sets ranging from 10 to 30 runs of each code, so multiple versions could run at the same time.

Each system allocated dedicated nodes to the tasks and had other work running on different nodes. All runs used a one-to-one mapping of CPUs to tasks, which meant that the nodes were fully packed and all CPUs were used. Table 7-2 summarized the results for the primary test runs.

The main point of the table is that some systems have larger variability than others. In the T3E's case the variability stems from migrating jobs due to scheduling. On the other three systems consistency greater than 95% and in some cases almost 99% for parallel jobs running in a full production mode is achievable.

System		EP	LU	FT
	Number of Runs	70	119	118
Cray T3E	Mean Runtime (sec)	35.5	305.2	106.5
	Standard Dev (sec)	2.2	47.8	12.1
	Coefficient of Variance	6.11%	15.58%	11.33%
	Number of Runs	424	165	210
IBM Power 3+ SP (Seaborg)	Mean Runtime (sec)	17.4	74.6	41.5
	Standard Dev (sec)	0.09	3.4	2.4
	Coefficient of Variance	0.52%	4.58%	5.70%
	Number of Runs	336	359	371
Compaq SC	Mean Runtime (sec)	5.03	42.8	30.6
	Standard Dev (sec)	0.35	1.9	1.0
	Coefficient of Variance	6.91%	4.53%	3.18%
	Number of Runs	112	71	119
Intel Cluster	Mean Runtime (sec)	17.6	408.7	90.7
	Standard Dev (sec)	0.03	10.7	1.0
	Coefficient of Variance	0.17%	2.62%	1.07%

Table 7-2: Shows the basic statistics for the test runs. Including some of the special test cases discussed below, over 2,500 test runs were made. There was no correlation between which nodes were used and the performance or consistency on the system.

7.6.4 System Configuration Issues

Often consistency issues can be detected by comparing application runtimes. It is also possible consistency issues are masked when they impact a large portion of the resources on a system, so that all applications are in turn impacted. In this case, micro benchmarks are needed to identify inconsistency.

An example of this occurred on early Power 5 (Sinharoy, et al. Jul-Sep 2005) systems that were delivered in late 2005 and early 2006. Testing of the NERSC Power 5 system, Bassi, showed that there was approximately 30% difference in memory performance between even and odd numbered nodes. Figure 7-6 shows the memory performance within all the nodes in a rack on the Power 5 using the stream micro benchmark. The pattern of the odd nodes being faster than the even nodes is obvious and occurred regardless of the rack tested.

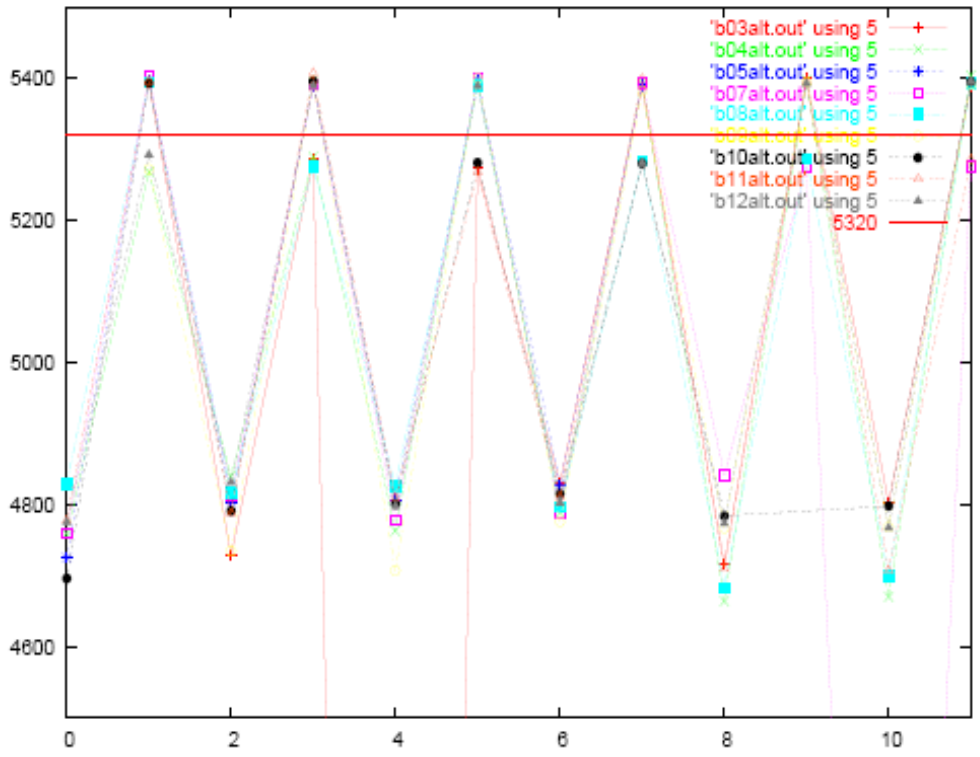


Figure 7-6: The Stream Memory rates for different nodes within a rack of the Bassi System. The Y axis is the memory rate reported from the Memory stream micro benchmark and the X axis is the node number within the rack. There are multiple runs of the same test used.

The problem was traced to a boot time memory allocation. Power 5 systems use the IBM's Federation High Performance Switch (HPS) and the switch interfaces can plug into the right or left side of the 8 core node. At the time, the system was using boot code that was written for the IBM Power 4 systems without modification. The boot code placed the initial memory page assignments in different locations based on the switch adaptor locations. On the Power 4, this had no impact on performance, but on the Power 5, it clearly did.

Figure 7-7: Picture of the IBM IH nodes – showing the location of the Federation interface alternating from left and right sides of the node. This simplified cable management for maintenance.

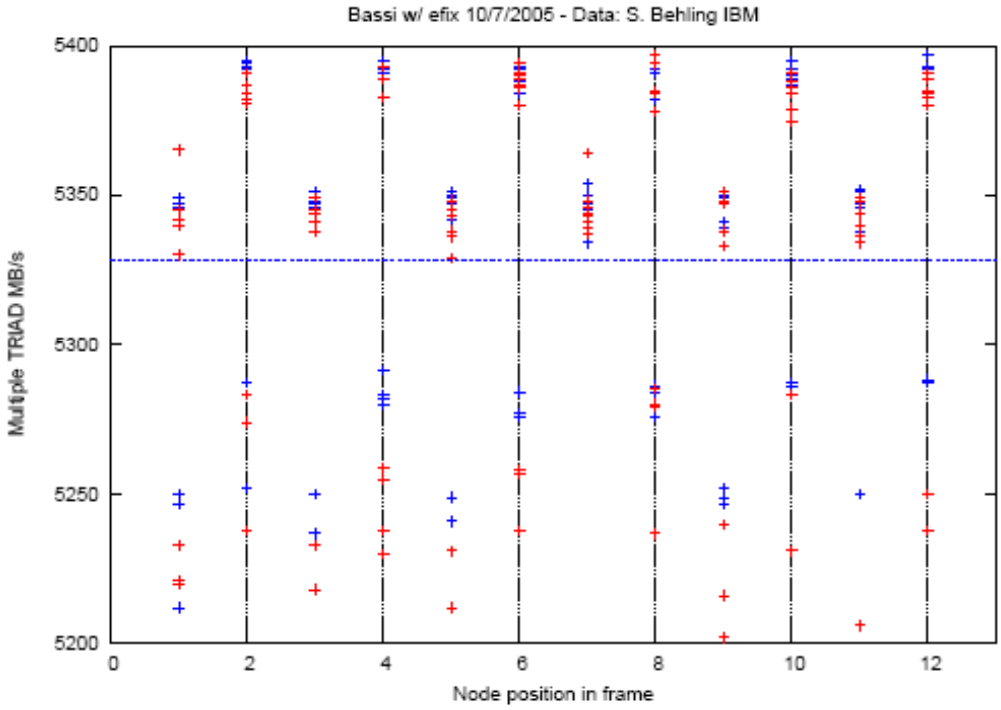


Figure-7-8: The same tests as were done in Figure 7-6 after the memory allocation routine was changed to place the initial large pages for system software in the same location. While there is still some natural variability in these different runs, it is much more consistent.

Figure-7-8 shows improved performance after the change and average application performance also improved. This inconsistency did not exhibit itself in parallel application performance since it was highly improbable that an application would be assigned to all even or all odd nodes. Since applications were always run on a mix of nodes, the slower memory performance on the even nodes dominated and caused a general application slow down.

7.7 Effects Of The Time of Day

Consistent runtime may depend on when a job is run. Inconsistency could be caused by system issues, such as the scheduling of large jobs, system

diagnostics, system management activities (e.g. collecting accounting records, files system backups, etc.), or daemons, as well as by user activity, which may peak at certain times. Analysis showed that for the runs on the IBM SP and Pittsburgh Supercomputer Center's Compaq systems, time of day had no significant impact on consistency. There were not enough runs on the T3E or the Intel cluster to perform this analysis for those systems in a rigorous manner, but no patterns were observed to contradict the results on the SP and Compaq.

7.8 Embarrassingly Parallel (EP) Consistency

Two machines, the T3E and PSC Compaq, showed unexpected inconsistency for EP runs. As its name suggests, EP does very little communication. However, because it has a relatively short runtime, it is possible individual cases of network congestion caused this inconsistency. If a version of EP that does not use the network still shows significant inconsistency, the individual node must be at fault. To test this theory, single node versions of EP were run on the T3E with one core per node and on a four-core per node Compaq. The coefficient of variation for the Compaq dropped to less than 1.6%, indicating that something on the node was causing some inconsistency. CoV dropped to 1.5% on the T3E.

The NPB codes report their own timing and performance rate. In order to do that, they use a standard system call to get the relative time between the start and the end of the program run. The T3E compute nodes run a very

limited microkernel that uses system call forwarding. While many system calls are forwarded to service nodes, timing calls run on the local processor element, which in this case would use the timer in the rank 0 MPI task. The T3E's global clock was used to keep time and clock interrupts aligned between the PE (Private Communication with Mr. Steve Luzmoor, Cray Inc. 2008).

When only CPU time was measured on the T3E and the Compaq, CoV was less than 0.5%. From this, we can conclude that the network and, in the case of the T3E, the NPB method of timing, were responsible for most of the inconsistency. Only on the Compaq system do individual nodes contribute somewhat to the inconsistency.

7.9 Changing the Number of Adapters

Two machines, the IBM SP and the Compaq system, have a variable number of network ports (adapters) on each node. It is logical to expect that using more adapters would decrease runtime and improve consistency for the LU and FT benchmarks, but not for the EP benchmark with its trivial communication requirements. A set of test runs was made of all three benchmarks using both one and two adapters.

Changing from one to two adapters had a statistically significant effect on mean runtime only for the EP and FT program runs on the IBM SP machine ($p < .01$ in both cases). For statistical significance, p being lower than the significance value, α, means there is only a 1- α probability the conclusion

significance is incorrect. Using an F-test (Selby 1968) to compare consistency changes shows two adapters decrease consistency for the FT benchmark on both systems ($p < .05$ in both cases). Consistency for the LU benchmark increased with two adapters on both the Compaq ($p < .01$) and the IBM SP ($p < .01$). As expected, changing the number of adapters used had no significant effect on the EP benchmarks.

Increasing the number of adapters did not have much impact on mean runtime, probably because the test programs did not send enough data to benefit from an increase in bandwidth. Analysis of Variance (ANOVA) indicates that changing adapters had the following effects on runtime:

a) no effect on PSC ($p > .1$)

b) no effect on SP for the LU code ($p > .1$)

c) an effect on the IBM SP both the EP and SP codes ($p < .01$)

7.10 Low Consistency on the Cray T3E

With the exception of the T3E, all of the machines studied had distributions such as the one shown in Figure 7-5. In essence, the distributions were normal bell curves with long right tails. Almost every code experienced some normally distributed slow down, while a few suffered significantly more. A typical distribution for the T3E is shown in Figure 7-6. The majority of jobs experienced only a very small slowdown, while a significant portion suffered a far larger slowdown (40% or more in some

cases). To understand this, the system logs for the runs were examined and only the system time was measured. This caused the histograms to collapse into ones similar to Figure 7-5, as seen in Figure 7-7.

This higher level of consistency corresponded to the users experience on the T3E. The users, who paid close attention to their charged time for each of their runs, felt the T3E was very consistent.

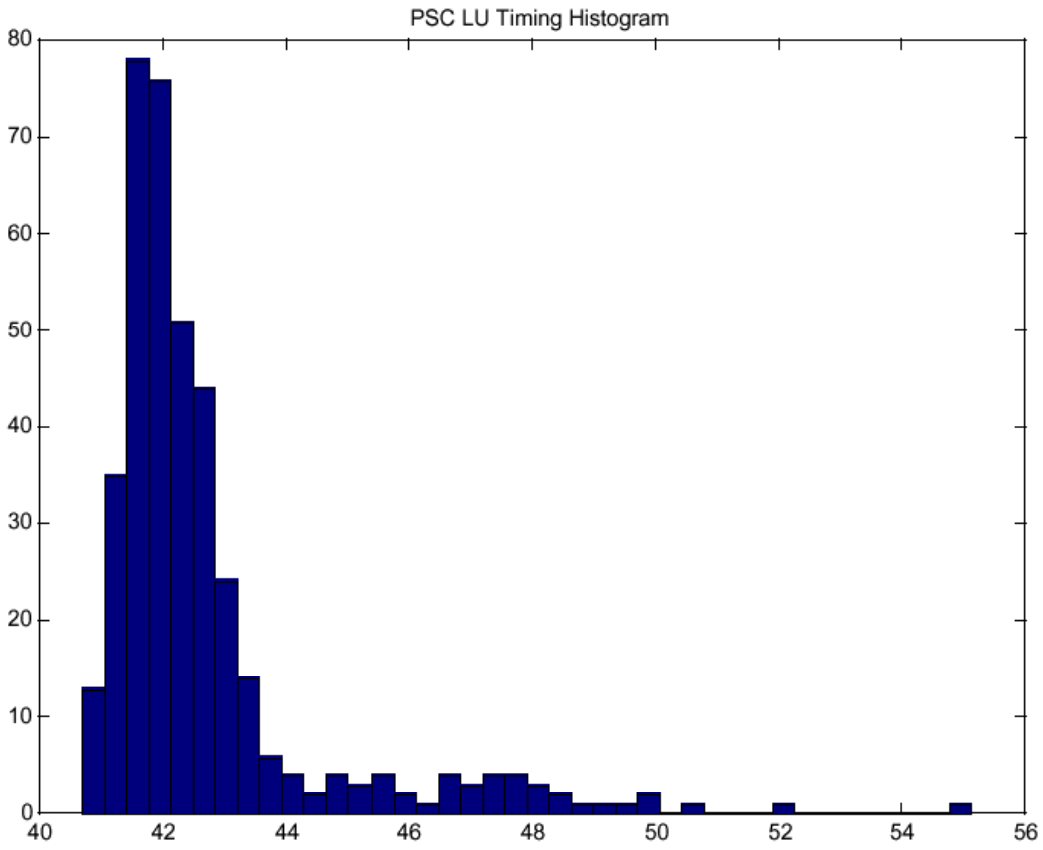


Figure 7-9: This is a histogram of LU times from the Compaq SC system. It shows Gaussian distribution with a long fat tail for runtimes.

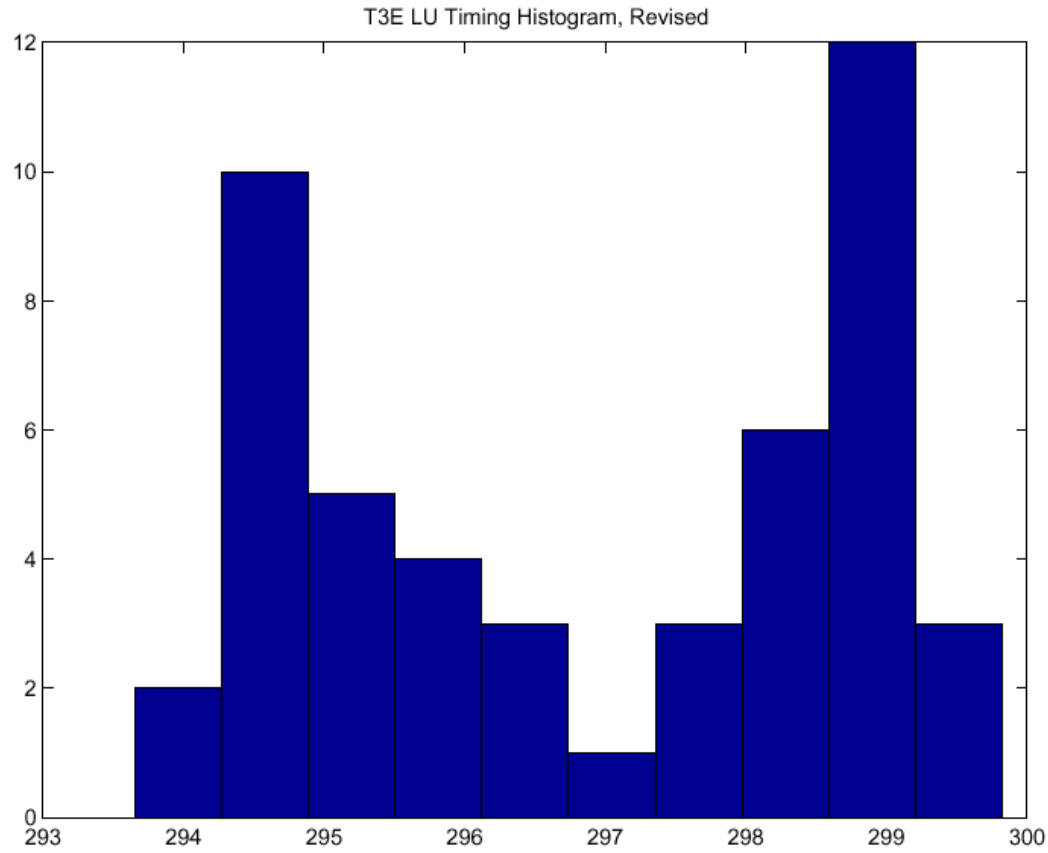


Figure 7-10: This is the histogram of the LU times on the T3E system. It shows a bi-modal distribution with a large range for the runtimes.

Lack of consistency indicated for the Cray T3E was surprising and investigated further. In order to make efficient use of the Torus network (Cray Research, Inc. 1996), the T3E assigned logical node numbers to physical PEs at boot time (Cray Research, Inc. May 1997). Physical node numbers are based on how the node physically connects in the interconnect network. Logical numbers are assigned deterministically to minimize routing at system startup. The T3E interconnect does direction ordered routing and special routing, but adaptive routing was never implemented, so the T3E only routed the data through a predefined path using virtual channels.

Jobs are scheduled on logically contiguous nodes. This means that large contiguous blocks of PEs (PE is the T3E terminology for which other systems call a node. Each PE has one CPU core) gradually become fragmented, making it increasingly difficult to run jobs requiring large numbers of CPUs. The T3E addresses this limitation by periodically scanning all the PEs and identifying ones that have no work assigned. In a manner similar to memory shuffling, the system “migrates” jobs amongst nodes to pack all the running PEs together. This creates larger sets of logically contiguous PEs for new jobs to start. Jobs are assigned to PEs using a number of parameters, including an alignment measure that indicates how the starting point and/or ending point of the application aligns to power of two logical PE node number (Cray Research, Inc. 1996).

In order to efficiently schedule new jobs, the T3E system software, called the Global Resource Manager (GRM), scans all the nodes to look for opportunities to migrate. The frequency of scans is site selectable; on the T3E under study, they occurred at five second intervals. Such frequency was set to balance the “competition” between the Global Resource Manager and the load balancing routines.

When jobs are migrated, system accounting adjusts the charged time to compensate for the time the job is moving and not processing because it is not assigned to a node. This is fair to users but over represents the consistency. However, the real time clock continues, which is what is used to

report the NPB runtimes. System accounting logs have been correlated with the output of the NPB tests. Table 7-3 shows the difference between the real time values and the system accounting time for the job.

Cray T3E	EP CoV	LU CoV	FT CoV
NPB Reported Runtimes using wall clock	6.11%	15.58%	11.33%
System accounting reported runtimes which did not count time spent during job migration.	0.8%	0.6%	0.93%

Table 7-3: Compares T3E consistency using actual NPB Runtime reports and system accounting data. The NPB runtime reports calculate the "wall clock" time for the test – and do not adjust for time lost due to migration or checkpoints.

Table 7-3 shows that, adjusted for time spent being migrated, the consistency of the T3E improved considerably. The situation caused by the T3E having to migrate to maintain a mapping of nodes to the location in the switch fabric has interesting tradeoffs. Some of the impact of inconsistency discussed earlier was mitigated with the adjustment in the accounted time, since jobs did not abort due to exceeding the runtimes. Yet there were consequences, such as, users waiting longer for results. Not migrating also has consequences, since certain work will not progress through the system in as timely a manner and system productivity will decrease. Infrequent mitigations would likely lead to lower utilization, while too frequent migrations decrease consistency. Hence the tradeoff of the frequency of job mitigations to the consistency should be assessed for each workload to reach a balanced solution.

7.11 Detecting and Reacting to Inconsistency

Off-line detection and reaction to inconsistency is possible. The first sections of the chapter provide example methods system managers and developers use to drive inconsistency down into single digit CoVs. Unfortunately, sometimes this effort is large and requires a broad range of expertise. The work discussed below for the IBM SP required a team of 12 experts working together for 6 months, having skills in such areas as switch software and hardware, operating systems, MPI, compilers, mathematical libraries, applications and system administration. The improvements also required major modifications to the switch micro code, lowest level software drivers and global file systems.

Detecting inconsistency in real time so an application can respond is difficult to do. Dynamically detecting and responding in the proper manner is even more difficult. Some codes, such as the Gordon Bell Prize winning *LSMS* (Ujfalussy, et al. November, 1998), are internally instrumented to report the performance of internal steps – such as reporting the overall performance or length of time taken for a time step of simulation. From there, it is feasible to consider monitoring the periods and identifying if the past period is within an appropriate range, but this is not normal.

7.11.1 What To Do When Inconsistency Is Detected

Even when inconsistency is detectable, it is not clear what the proper action to take would be without a better model of what is going on. For

example, it is unclear whether adding more CPUs to an application improves consistency, because it changes network traffic patterns and forces the application to scale more. Likewise, it may be that decreasing the number of CPUs the application uses would improve the consistency and possibly performance.

Several teams use large shared codes and they run special tests on target systems to assess performance tradeoffs before setting the parameters for their codes. Other codes use auto-tuning based on system features. In these cases, it may be useful to add tests that focus on consistency as well as performance. However, it is uncommon for people to also test for tradeoffs in consistency. Another concern is the fact an application spending time monitoring and deciding what to do, will have a longer runtime and possibly create or even generate inconsistency in performance.

7.12 System Activity

In some cases, the root cause of a performance inconsistency can come from outside of the node, switch, and application. Most large parallel computers have a higher level control layer for system health, power, and connectivity monitoring. Such layers are designed to be all but invisible to applications, but unfortunately they can exert an influence on application performance.

7.12.1 Improving Consistency

The IBM SP at NERSC was doubled in size in early 2003. Due to previous experience with inconsistent performance, great care was taken to assure all known causes of variability were eliminated. Some of these included:

- All new hardware was identical to the existing hardware.
- In order to assure the software configurations between the additional and original nodes were identical; an exact image of all existing software was made and copied onto the new nodes.
- All system administrative procedures were identical.

In order to enable testing, new nodes were separated into a queue used for testing and an early user program. During this test period, benchmarking with a variety of applications showed that the new nodes were performing better - approximately 10% - than the original hardware. Figure 7-12 so an example of the different using the NPB LU benchmark. Once the problem was detected, the NPB tests were used as probes because they show the difference and were simple and shorting running. All reports of inconsistency involved parallel MPI codes. No serial performance differences were detected. At other times, the additional and original sets of nodes showed very similar parallel performance. After the new nodes were integrated with the existing nodes, comparisons of parallel jobs between original and additional nodes became difficult due to do IBM's *LoadLeveler* scheduling

implementation, but the difference in performance between new and old nodes continued.

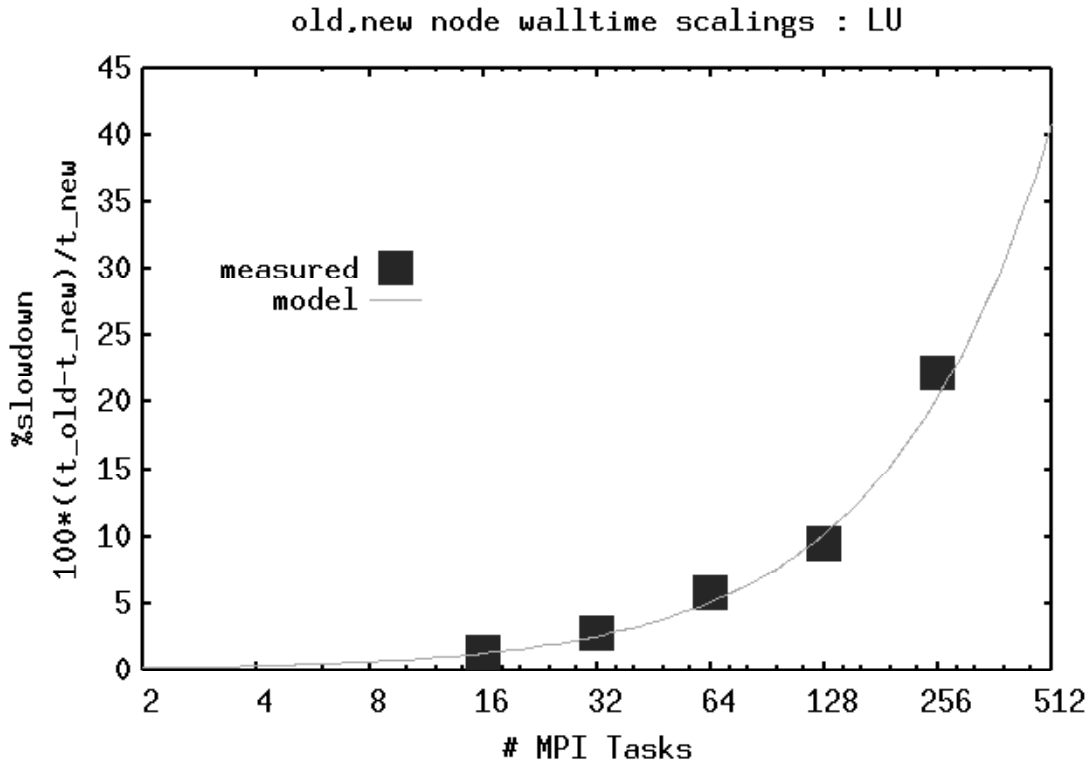


Figure 7-11: Comparison of measured and modeled slowdown between two sets of nodes in a parallel computer. The Y axis gives the relative slowdown between the two sets. The dark squares are measured performance data and the light line is a model describing the scaling of the slowdown.

The systems were audited extensively and no cause was determined until the events described in Section 7.12.4 below. The auditing included review of all hardware components for version levels, manufacturing, and other criteria. All software was verified for versions and parameters on each node.

7.12.2 An Observation Related To Concurrency

Over time, a body of data and timings established that parallel jobs ran slower, in proportion to their concurrency, on the original nodes. The degree

of the difference depended on the concurrency and the amount of synchronization in the MPI calls used in the code. A test case employed in the resolution of this issue was the NAS parallel benchmark LU because it was turned out to be a fast, reliable probe that coincided with the performance difference of full scale applications.

Since serial codes show no measurable difference, the parts of the parallel codes that involved synchronization were implicated. Interruptions at the OS level or at the switch adapter level can have a minimal impact on serial processes, but compound when many concurrent processes are interrupted. If a linear model of frequent short interruptions on each node is extrapolated, the old nodes have half the performance of the new nodes (for LU decomposition) at a concurrency of 1250 tasks! Obviously, this is a significant impact on user productivity. Everything observed from the testing showed that synchronization of parallel jobs was impacted by delays proportional to concurrency. However, it was not known if these interruptions were from hardware or software.

7.12.3 Steps For Diagnosing The Problem

Diagnosing this problem took a number of months, with the system mostly in production mode most of the time. The steps that were followed were:

- Application testing: By profiling application runs on separate collections of original and new nodes - including segmented switch sub trees – it

was determined which sections of the code account for the overall timing differences. Knowing the time spent in each MPI routine showed that the variation was related to a small number of MPI functions. For the NPB LU code, nearly all of the asymmetry in wall clock time occurred in the MPI routine MPI_Wait. Since the MPI software on all nodes had been shown to be the same, the MPI_Wait differences were the result of something outside of the scope of the application and/or MPI libraries. This testing only indicated that MPI_Wait took longer to synchronize codes on original nodes than it did on new nodes.

- Hardware testing: IBM pursued the issue from the point of view of hardware differences. A plan was developed and carried out to swap hardware components between old and new nodes in order to identify hardware that might be responsible for the observed asymmetry. IBM sent hardware engineers out to complete the hardware testing. After the changes were made, the NPB LU benchmark was run. CPU's, memory books, system planes, and switch adapters were all swapped between the two sets of nodes without observing a change in the timings, indicating the issue was not based on a hardware difference between the old and new nodes.
- OS testing: NERSC staff double checked that the OS images used to install the batch nodes were uniform. Using identical images for OS install was part of the standard methods for system administration of

Seaborg. Checksums of system libraries were compared and no asymmetries were found.

7.12.4 The Problem Was Found – The Control Work Station

After months trying to identify the causes of variability, the reason was uncovered through a combination of luck and observation. During one of the evaluation periods of running tests to study the different performance, the IBM SP Control WorkStation (CWS) for the system failed. The IBM SP is designed to continue operating without a CWS, and it fortunately did. Unexpectedly, while the CWS was down, jobs ran very consistently regardless of whether they used the original or the new nodes. When the CWS was restored to service, inconsistency was observed again. Figure 7-9 shows NPB LU runs that happened at the point of the CWS failure.

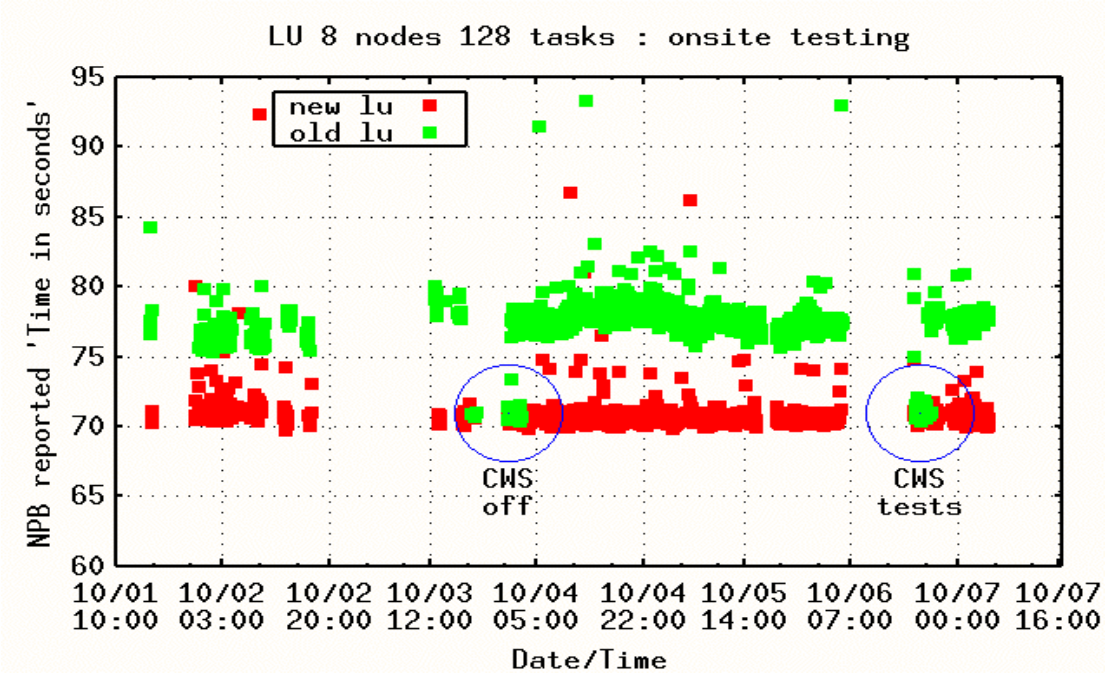


Figure 7-12: Runtimes of the NPB LU on original ("old") and additional ("new") nodes. Note how the time on the old and the new nodes are essentially the same when the CWS was off and then later when there was CWS testing to resolve the issues.

Process Name	#calls original	#calls additional	Asymmetry ratio
#spget sy	191817	6864	27.945367
#fcistm	192121	7188	26.728019
#lssrc	194608	7780	25.013882
#basename	385701	15550	24.803923
#odmget	193918	8481	22.864992
#ksh	390625	20129	19.406081
#rm	197514	12449	15.865853
#sed	397999	29482	13.499729
ksh	206206	23159	8.903925

Table 7-4: The difference frequency that different system reliability tasks run on the original and additional nodes.

With this hint, UNIX process accounting logs showed the CWS was running several processes and other system administrative commands up to 27 times more often on certain nodes than other nodes. Knowing the specific processes and their frequencies provided a fingerprint of the subsystem

causing the interruptions. Finding command names occurring in the system script /usr/lpp/csd/bin/ha.vsd which was invoked as part of the First Failure Data Collection subsystem, led IBM to investigate the CWS problem management subsystem, *pman* (IBM, Inc. n.d.). The analysis determined that four definitions were deactivated in the system management GUI (Graphical User Interface), but were still running. This was a bug in the *pman* subsystem data base that was not observable to system managers through the GUI. Instead, the definitions required explicit deletion from the System Data Repository (SDR which is a data base of the system configuration) to remove them, rather than deactivation as was documented. A problem report was opened with IBM development (PMR #38446) that corrected the defect in the problem management subsystem, while the workaround was to actually delete the files.

Once the definitions were deactivated, applications ran with very low variability regardless of how many new and old nodes it used. Resolving this issue led to a measurable improvement for synchronizing MPI codes at high concurrency. In normal operation, jobs use a combination of old and new nodes. Thus, the end result was that all codes saw a benefit of faster and more consistent runtimes, particularly those codes at higher concurrency. The performance gains shown earlier in Figure 7-2 and Figure 7-3 are a result of the changes described in this section and were realized due to consistency testing.

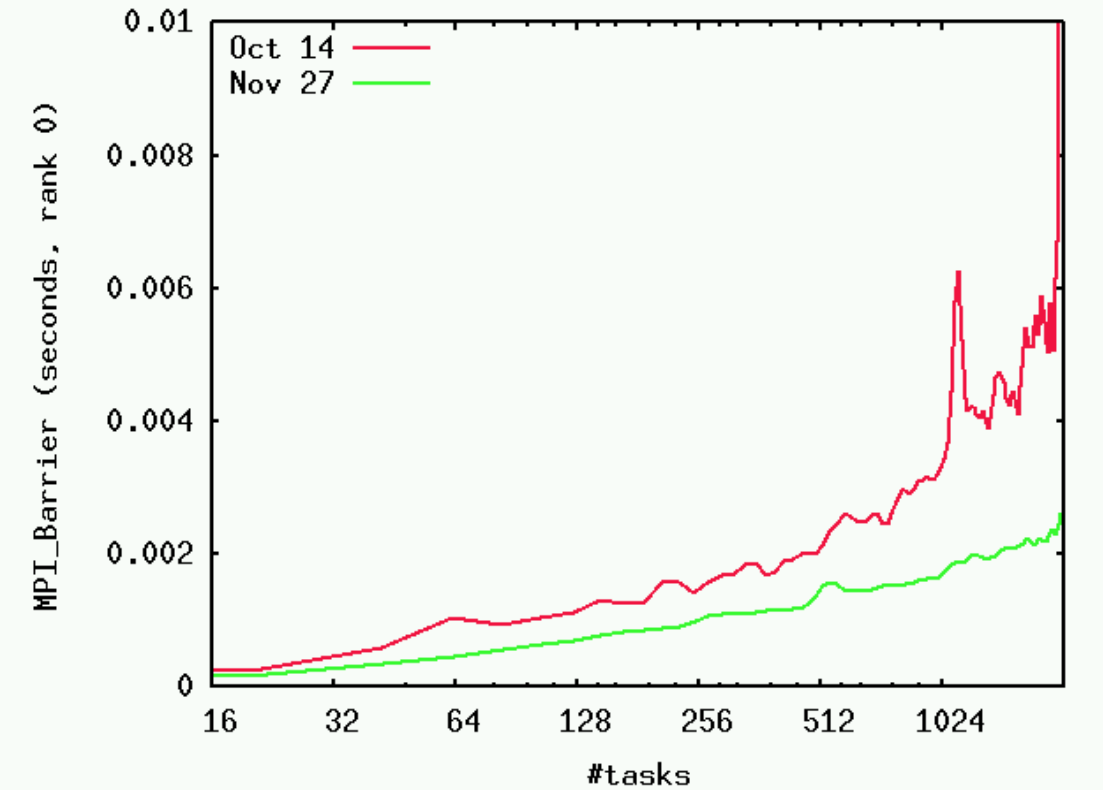


Figure 7-13: The above graph shows MPI barrier performance before (Oct 14, 2003) and after (Nov 27, 2003) the problem was corrected. The graph is smoothed to make the trend clearer. The lower barrier times after the problem resolution led directly to improved performance for many applications.

7.12.5 An Observation About Performance Variability

This case study also demonstrated that when the time scale of performance variability becomes short enough, it manifests itself as a static performance penalty for all applications. The resolution of this issue was the equivalent to providing the users a ½ TFlops/s peak system* for no cost other than diagnosis effort.

* $Potency_{IBM} = Potency'_{IBM} + .1*Potency'_{IBM}/2 = 1.05 * Potency'_{IBM}$

7.13 Chapter Conclusion

HPC systems increase complexity, both in their hardware and software layers. As complexity increases, there is increased opportunity for system attributes to contribute to inconsistency for the time it takes the system to complete a given amount of work. Some inconsistency is to be expected as different applications compete for shared resources. This inconsistency is unavoidable, but can be minimized to a few percent CoV with good system management and work scheduling.

There are systematic contributors to inconsistency that can be avoided. The contributors can be from system administration, software and hardware. The “low hanging” fruit which can easily be rectified include having system nodes with different software levels and allowing unnecessary system processes to execute. Even after these causes are eliminated or mitigated, many areas can contribute to inconsistency of a system.

The work in this chapter shows hardware, configuration management and software each contribute to observable inconsistency. It shows well configured, error free large systems are capable of consistently executing a wide range of applications simultaneously.

Chapter 8: Usability – Something for the Future

8.1 Chapter Summary

Usability is the final piece of the PERCU methodology and has been a rich research area in its own right for a long time. Assessing usability for HPC systems will take multiple dissertations. Indeed, the Defense Advanced Research Project Agency (DARPA) is funding significant "productivity" research in HPC as part of the DARPA High Productivity Computing Systems (HPCS n.d.) effort. Computer Human Interfaces are a complete discipline with large organizations such as the Association of Computing Machines Special Interest Group on Computer-Human Interface (ACM SIGCHI) involved with improving productivity, but seldom is the work targeted to HPC systems. Similarly, Software Engineering is a discipline that deals with usability by addressing improved methods of code development and maintenance (Boehm May 20-24, 2006) but is, in many ways, orthogonal to the assessment of usability of HPC systems. More recently, the field of Usability Engineering (ISO Usability n.d.) has emerged as well, as has the ANSI/ISO Usability Model (ISO 1998) – a standard the deals with "office work" and visual displays.

Given the degrees of focus from others in this area, this dissertation does not attempt to explore Usability in a HPC specific manner nor does it attempt to expand the field of usability analysis. Rather this dissertation documents existing "best practice" approaches within the HPC community to assess

usability. It provides references to new and ongoing work with comments on the likely impact for HPC system evaluation. It also presents two examples of using some of the common usability assessment approaches for comparative study of HPC systems.

8.2 Approaches to Assess Usability

What scientists really want to know of the usability of a system is "*How much harder it is to use an HPC system rather than standard platform tools?"* and *"Is it worth learning how to use a much more sophisticated and/or efficient system than a system they already know how to use?"*

Usability addresses these questions. (Nielson and Levy 1994) noted two fundamental approaches to usability analysis, while admitting "usability is a general concept that cannot be measured, but is related to … parameters". The two approaches are subjective user preference measures to assess how much users of a system like something, and objective performance measures that assess how "capable" users are of exploiting a system well. One of the conclusions they noted was a strong association with users' average performance on tasks related to using systems and the user's subjective satisfaction so "there is a large chance of success if one chooses … based solely on user opinion."

According to Pancake (Pancake June 1998), usability, at least for software interfaces, should be assessed for:

- Ease of learning (e.g. intuitive conceptual frameworks and consistency);
- Ease of use (minimal complexity);
- Usefulness (applying to new situations and avoiding errors); and
- Throughput (streamlining a sequence of operations, reducing errors, being efficient).

There are a number of studies discussing the ISO/ANSI usability model that defines effectiveness* (errors made), efficiency measures (time to complete a task) and satisfaction (preference).

(Sauro and Kindlund April 2-7, 2005) developed a method to simplify the ANSI/ISO usability metrics into a single metric that is based on the correlation they found across the usability assessment of 1680 tasks. They proposed a method, called Principle Components Analysis, to scalably provide a summarization model of usability and a single metric that can be used in regression analysis, hypothesis testing, and usability reporting. (Hornbaek April 28-May 3, 2007) did a study of over 73 usability analyses and concluded "correlations are generally low" between the usability measures used in the studies. The paper specifically found much lower correlation than claimed in

* Note usability terminology uses a different definition of *effectiveness* than the term used throughout this work to reflect the effectiveness of work and resource scheduling in computer systems.

the (Sauro and Kindlund, 2005) study and therefore questioned the idea of a single usability metric.

Much of the documented research on usability work fits into several categories. One approach is fundamental usability analysis investigations that focus on human interactive use of machines. Many HPC systems have interactive access, but the majority of the resources are consumed by non-interactive activities. Usability is an active area of investigation, but it may be too early to apply it to the evaluation of HPC system in the context of comparison.

(Dicks October 20-21, 2002) and (Rubin 1994) discussed four reasons why even rigorous forms of testing may not assure usefulness because testing is always in an artificial situation and does not prove a system works. Furthermore, usability testing participants are not fully representative of the target population of users and "testing is not always the best technique to use." (Dicks, 2002) further points out *ease of use* and *usefulness* are often used synonymously, but they are very different. Ease of use indicates how quickly a user can use a system to complete a task and is related to efficiency. Usefulness is the overall usefulness of the system – does it do what it is supposed to do.

A more common approach, one used almost always in HPC, is comparative studies between systems. These studies typically assess systems in a manner that includes both qualitative and quantitative criteria.

Several examples exist of this approach and are discussed below. More recent HPC efforts associated with usability deal with the broader area of productivity. One currently underway is the effort to assess improved productivity for the DARAP HPCS program and is discussed in Section 8.3 below. Another discussion about productivity across the entire system, based on economic theory is included in Appendix E.

8.3 High Productivity Baseline Studies

(Gould and Lewis March 1985)recommended three principles of designing systems: early focus on user and tasks, empirical measurement, and iterative design. The DARPA HPCS study is assessing the relative productivity of systems at two different points in time, mostly by addressing the first and second principles. This study, underway at IBM and Cray Inc, assesses five use cases: code development, code maintenance, production running, system management. The code development category is further separated into single user versus team developed codes to make five cases.

NERSC and the author of this dissertation are collaborating with IBM in three areas of the study – individual and team code development, and system management. The goal of this study is to document usability inhibitors for systems in the 2002 time frame in order to design better systems that will be available in the 2011 time frame. This information will then be used to guide software and system improvements for the IBM PERCS system scheduled for deployment in 2011 (Danis and Halverson February, 2006).

The current effort is to formalize study methods and gather data by electronically instrumenting single user development tasks that ask subjects to create parallel programs to solve a particular problem. The subjects are selected for the range of parallel programming experience they have. Their progress and approach are monitored. Interviews are done to assess the problems and issues they felt were problematic. This effort is in the data collection phase and reports are expected later. Most of the data collection is being done on NERSC's *Seaborg* system and also on the NERSC *Bassi* system, both parts of the IBM Power series of systems.

The other effort associated with this project is to assess system management productivity. The approach was to interview professional HPC system managers and to estimate the time spent in different tasks such as system upgrade preparation and execution, problem diagnosis, job scheduling monitoring and adjustments, and file system management. The discrete tasks for these functions are being broken down and assessed. The amount of work in each area is also being analyzed along with failure and rework information. This information will be used to identify areas for improvement in future systems and then eventually to measure the difference between the systems of 2002 and 2011.

8.4 Comparative Usability of Two LWOSs

As discussed in Section 8.2 above, comparative study of systems for usability assessment is appropriate. Section 8.3 above discusses long term

assessment of similar systems over a long term basis to identify and validate design changes. System assessments that use PERCU are more often carried out on roughly contemporaneous systems for the purpose of comparing the relative potential and value of systems. Hence, a common approach is to compare two or more systems across a set of functions and activities. This section provides a specific discussion of carrying out just such a comparison.

8.4.1 Comparison of CVN and CLE

Rather than to discuss a comparison in the abstract, it is instructive to discuss the relative usability evaluation of the two Light Weight Operating Systems – CVN and CLE. This is a limited analysis in the sense the criteria discussed above recognized both systems had limitations that impacted usability – at least at the time of evaluation. For example, neither system was expected to support many of the programming models found in general purpose systems.

Both CVN and CLE used in this study had full-featured programming environments ((DeRose and Levesque May 7-10, 2007), including:

- PGI, Pathscale, and Gnu compilers for Fortran, C and C++ codes;
- Cray's Portals communication layer that supports MPI and SHMEM parallel programming models;

- Cray LibSci and AMD Core Math libraries;
- Cray performance and profiling tools; modules environment for managing system and custom built software;
- Torque/Moab for batch system managements; and
- Lustre parallel file system.

8.4.2 Evaluation Criteria

Usability was assessed for both ease of use by the computational community and ease of management for system managers. 377 separate criteria were examined for CLE and CVN. Expected features were tested for functionality as well as performance. For CLE, of the 377 items, 254 were testable for CLE for this analysis – with 35 applying only to future functions and 88 being more descriptive and not testable. Similarly, of the 377 items, 248 were testable for CVN – with 94 being descriptive and/or not testable. For CLE, more than half, 53%, of these criteria were related to Usability, with 39% focused on Usability issues and 14% on System Manager Usability.

Table 8-1 shows the comparison of how many usability features were operational between CVN and CLE. Less than 10% of the items under CLE failed their tests. Almost all failures regard modest to slight discrepancies with performance. As of January 2008, only one of the 134 Usability tests is

currently outstanding for CLE - proper functioning of disk quotas which is currently a high priority problem report with IBM.

	CVN	CLE
Number of features tested	248	254
Number of features properly working	232-90.5%	232 – 91.3%

Table 8-1: Initial Usability Tests for CVN and CLE.

8.4.3 Observations

Usability was truly evaluated by moving large scale applications to the systems – which was qualitatively evaluated in conjunction with early users. The usability advantages of CLE over CVN are a bigger set of standard POSIX C library routines for compute node applications, so users have more control for their applications, and less need to rewrite the source codes. CLE's increased Operating System functionality simplified code porting from other platforms than CVN. At least in some cases, compilations are quicker. CLE provides other needed functions, such as OpenMP, pthreads, Lustre failover, and the possibility of adding Checkpoint/Restart and other features. CLE enabled more options for debugging tools, such as the Allinea DDT (Distributed Debugging Tool) (Allinea n.d.), which is now the operational debugger running on Franklin.

Some disadvantages of CLE over CVN are the increased memory footprint for the operating system so that it leaves less usable memory space for user applications. The difference is about 170 MB/node from our measurements (about 4.25% of the available memory). MPI latency for the

farthest intra-node is higher under CLE (8.12 μsec) than CVN (7.55 μsec), although this may improve for future CLE OS releases.

8.4.4 CLE and CVN Evaluation Feedback

NERSC launched an early user program on Franklin during the CVN and CNL assessment period. We worked with experienced users on Franklin to benefit all parties. Many early users were able to run high concurrency jobs to tackle much larger problem sizes and model resolutions that were impossible before. Users got a chance to get hands-on with a new architecture and a relatively lightly loaded system, and user jobs were free of charge from their allocations. Running the broader range of user applications helped find any problems (and fixes) in the system. The overall user feedback for CLE was very positive, even at its early exposure. Most applications were relatively easy to port to Franklin, the user environment (via modules) was familiar, and the batch system worked well.

8.5 User Surveys

Several studies noted in Section 8.2 above note user satisfaction is a good indicator of usability. Unfortunately, it is not always possible to use survey instruments to assess usability of HPC because there are limited communities in common across different systems and access may be limited for systems that are already deployed. More importantly, assessment of systems not yet deployed is not possible with surveys of user satisfaction.

Nonetheless, user survey instruments play a key role in guiding near term operations of HPC systems as well as longer term decisions based on the indicators within the survey. One of the longest running survey series available for HPC systems is at NERSC, which surveyed (NERSC User Survey 2006) a population of 2,000 to 3,000 HPC users over a period of 10 years.

The survey allows comparison on a side-by-side basis of user satisfaction for six different HPC systems. The parameters are not just usability, but asked users to rate all systems and services across a set of parameters using a scale from 1 to 7 (least to most satisfied). The survey asked users to rate the importance of systems and services as well as their satisfaction. Users were encouraged to provide free form comments, suggestions and recommendations as well.

In 2006, users were asked to rate each system at NERSC in the following categories:

- Hardware/System: Overall satisfaction, Uptime (Availability), Batch wait time, Batch queue structure, Ability to run interactively, Disk configuration and I/O performance
- Software: Software environment, Fortran compilers, C/C++ compilers, Applications software, Programming libraries, Performance and debugging tools, General tools and utilities and Visualization software
- An overall rating of the systems

Detailed analysis of the surveys can be seen at http://www.nersc.gov/news/survey/. From studying the results several conclusions are obvious.

- User satisfaction varies from year to year on the same systems.
- System load plays a major part in the impressions of the users.
- Satisfaction for systems generally improves over time, but there are several cases of a mature system degrading as it remains popular with users but not as well suited to the increased load.
- User support and help using the systems (e.g. advice, training, documentation, etc.) are key components to satisfaction.
- Areas that are persistently a concern are the ability for systems to schedule the user's work in some organized manner (batch wait time) and more computing and storage resources. This observation from the survey results and user input, directly led to the creation of effectiveness category in PERCU and the ESP test.

8.6 Chapter Conclusion

The current HPC practice of system usability assessment is mostly comparative analysis using checklists of features and services. There is little formal study of HPC usability.

User surveys are good at assessing deployed systems for operational improvement and for indicating important features that influence usability, but are seldom used for usability assessment of future systems. Other forms of

comparative analysis are underway to influence future system design – the largest being the DARPA HPCS program, but are long, large efforts that may not be applicable to system purchases or comparisons.

Chapter 9: PERCU's Impacts, Conclusions and Observations

9.1 Chapter Summary

This chapter discusses the impacts that PERCU and its components have had in the world of HPC for over the last decade. Several organizations have adopted parts of the PERCU method to supplement their methods. Several DOD Modernization Program (Private communication with Mr. Cray Henry, Director of the Department of Defense (DOD) High Performance Computing Modernization Program (HPCMP) 2008) sites (Army Research Laboratory, DoD's Engineering Research and Development Center (ERDC) are using the SSP concept to assure their systems are operating at the performance levels expected throughout their life. Lawrence Livermore National Laboratory considered using the ESP test in their ASCI procurements (Private communication with Mr. Brent Gorda, LLNL 2005). Consistency is a measure found in recent National Science Foundation RFPs. This chapter looks at each of the parts of the PERCU method, summarizes it, and comments on their impacts and what might lie in the future for that area.

9.2 Summary of PERCU

PERCU is a holistic, user based methodology to evaluate computing systems, which, in this work, is applied to high performance computing (HPC)

systems. It consisting of five components: Performance, Effectiveness, Reliability, Consistency, and Usability.

The Sustained System Performance Method is introduced for an improved way of evaluating systems for the potential to support a given workload. SSP was defined and several examples of its use provided. It can evaluate systems based on time-to-solution of a suite of tests and/or with price performance. SSP can take into account systems that are available at different times and also can be used throughout the life of a system to monitor performance.

This work also introduces the Effective System Performance Test that was developed to encourage and assess improvement for job launch and resource management features of systems – both important aspects for productive computing systems.

Reliability is the third characteristic of a productive system, and this work explores the major causes of failure for large systems and suggests improved methods for *a priori* assessment of the potential reliability of HPC systems.

Consistent execution of programs is a metric that is often overlooked in system assessments, but if lacking, can negatively impact a system's quality of service. This work provides background of why consistency can impact quality of service, what causes inconsistency, and it defines approaches for assessing it.

This work discusses usability, providing a discussion of common best practices and their ability to help evaluate systems. It also is a potential area of future work in how usability analysis can be better applied to HPC systems.

PERCU can be used, in all or part, and with a wide range detail and effort. At its simplest, it provides a framework to consider holistic evaluation of large systems. In its detail, it introduces a set of new measurement methods. The use and impact of each component is documented for multiple systems, mainly using systems that have been evaluated at the NERSC Facility. PERCU has been used extensively by NERSC. Other sites are also using the framework and the software components. The methodology will help many organizations to get better performance, be more effective, give users a more reliable system, be able to measure consistency, and be able to make the systems have improved usability. There have already been many significant impacts by sites that are using the methodology and its components. The chapter also lists some ideas for further study.

The world of HPC is expanding daily. It is my sincere hope that the work here will contribute to the effort to get systems that can better solve the world's complex problems.

9.3 The SSP Method for Assessing Performance

9.3.1 Summary

The key contribution of this work for Performance evaluation is the Sustained System Performance (SSP) concept which is a method that uses time-to-solution to assess the productive work potential of systems for an arbitrary large set of applications. The SSP provides a way to fairly compare systems which may be introduced with different timeframes and also provides an exact method to assess sustained price performance.

The dissertation defines the equations for SSP and provides a theoretical basis for the framework, while it also uses a few simple examples to provide the motivation for use of and implementation of SSP. SSP also enables time-to-solution for different application domains to be compared across systems to determine cost performance and value.

9.3.2 Impact

SSP has been successful in evaluating and monitoring the NERSC systems. The method has been deployed by several other HPC sites (e.g. Army Research Laboratory, DoD's Engineering Research and Development Center (ERDC)). Others are considering using SSP in the future. Multiple vendors express a preference for SSP as the evaluation factor. For example, Dr. Margaret (Peg) Williams, Vice President for Development at Cray, Inc. said (Williams 2007)

"The SSP metric is much more representative of actual system performance than other widely used metrics, such as LINPACK or HPL. If more HPC sites used this type of metric, they would procure well-balanced systems that performed well over a range of applications and there would be fewer surprises with the delivered systems".

9.3.3 Further Work

It is important to keep any performance evaluation approach vital so the rapid evolution of hardware and software does not make the test obsolete. The NAS Parallel Benchmarks have had five or six versions over the past 15 years. Therefore, while the SSP method has long term viability, it will have multiple versions that need to be created and tracked. The author believes, now that the latest version of the SSP-5 suite is openly available to the research community, it will become a meaningful replacement for simplistic or obsolete measures. The SSP-5 suite is available for download at:

- http://www.nersc.gov/projects/procurements/NERSC6 and
- http://www.nersc.gov/projects/ssp.php.

9.4 Effectiveness of Resource Use and Work Scheduling

9.4.1 Summary

Effectiveness is a component of PERCU method that assesses the ability for users of the system to efficiently access the performance they need in the system. To measure effectiveness, a system utilization benchmark, the Effectiveness System Performance (ESP) test, was developed as part of PERCU. A brief description of how ESP evolved is provided along with the

design goals for the test. Chapter 5 and Appendix G summarize uses of ESP to evaluate different job scheduling software packages.

9.4.2 Impact

ESP has been successful in multiple aspects. It evaluated batch systems on the same hardware and produced insights into batch system features and benefits. These insights led to significant improvements in several products now serving the HPC and other communities. ESP helped motivate and evaluate the design features of IBM's *Loadleveler* and the Cray's resource manager, as well as helped tune Cluster Resources' *Moab/Torque* job scheduler for the Cray XT-4 at NERSC.

9.4.3 Future Work

The ability to schedule work on Petaflop and Exaflop systems will need new functionality to effective schedule and manage workflows for reasons of scale and future application complexity. The design of improvements needs metrics for guidance. ESP provides that and can evolve as the scale of systems evolve. ESP would benefit from changes to allow better assessment of scheduling for unique architecture features, topology aware scheduling and for being able to extract diagnostic information from resource management software.

ESP is packaged as a freely available software archive, with facilities for simple installation and execution. It is located at http://www.nersc.gov/projects/esp.php.

9.5 Reliability

9.5.1 Summary

Reliability is the next aspect of system productivity. It is challenging, yet critical; to proactively assess the reliability of a system at the time of procurement before the system is purchased. This work documents causes of failure spanning six major HPC systems over five years. It identifies the major reasons that systems fail and shows that, at least for the systems included in the study, system wide outages were more often caused by software than hardware.

Chapter 6 discusses the reasons individual jobs fail on one system, and discusses improvements that resulted from analysis. The chapter suggests ways to improve the up-front potential for a system to be reliable as well.

9.5.2 Impact

A system reliability data repository has been created and contains six years of data on the NERSC systems. It is publically available as part of the Petascale Data Storage Institute SciDAC research collaboration. The web site allows interactive queries, charting and exporting of the data to CVS formatted files and is located at http://pdsi.nersc.gov.

Several sites including CMU (Carnegie Mellon University) and Los Alamos National Laboratory and others are using the system reliability data repository which contains six years of data on NERSC systems. It is the counterpart of the Computer Failure Data Repository, the first “publicly” distributable HPC failure repository.

9.5.3 Future Work

More can be done to fully understand the factors influencing reliability. The ways to diagnosis and improve software error propagation in the layers of HPC software is a key area of future work. Furthermore, the best, or even feasible, methods of doing up front comparison of systems for their potential reliability are very difficult. But as systems increase from 1,000s of processors and millions of components to 1,000,000s of processors and close to billions of discrete components, reliability, error analysis and error recovery will become paramount to the impact these systems will have. Application of methods like statistical learning theory and kernel machines have made some noticeable impact in other areas such as web services, but are not deployed in HPC in any wide degree.

Finally, single applications will have to continue to expand in scale as systems do. The ability to detect, accurately assess, and correct, single application failures will increase in importance. Once the reason for failure is accurately detectable there is another body of work to determine how best to

help applications recover, given the state of an application involves many software layers, not all of which are visible to the programmer.

Some of these factors have informed the requirements in the procurement for NERCS-6 and the experience gathered may help define how to improve proactive measures of reliability.

9.6 Consistency of Performance

9.6.1 Summary

It has also been shown that very large systems can be made consistent and the loss of cycles is avoidable for properly configured, designed and managed systems to the degree that consistency can be less than a few percent. Nonetheless, inconsistent performance can erode a system potential with constant vigilance. This work introduced the simple but effective Coefficient of Variation (CoV) metric that is used with a variety of benchmark testing to assess consistency. It helped to provide measurements that led to diagnosing issues related to inconsistency as shown in the real world examples related to consistency and how improvements to the system were made based on the metrics.

9.6.2 Impact

Some early sharing of these results has provided increased awareness about consistency. Several vendors have improved their software architectures based on the work documented in this dissertation – in addition

to fixing the discrete bugs that were identified with the work. Examples include improved job scheduling software with features such as checkpoint restart (IBM and Cray) and job preemption (IBM, Cray and LSF), better MPI launching methods (IBM), better consistency (IBM, Sun) and This resulted in more productive systems for users not just at NERSC but on systems at many different locations.

9.6.3 Future Work

It would be useful to add more systems to the study so there is more overlap of architectural features. This will narrow down features that influence performance consistency.

Studying consistency on more systems with the same architecture would show the impact of local configuration and system management choices.

Another step is to evaluate consistency using different variants of the same basic system architecture – for example, systems that have a different number of processors per node or that have the same CPUs with a different switch.

Another potential area to study is performance consistency differences between shared memory systems and distributed memory system. One may think consistency is lower in shared memory systems, but it is not clear that is the case, particularly on systems with more than 128 CPUs, such as the SGI Origin system (Laudon and Lenoski February 25-28, 1996).

Using specialized tests to perform finer grain studies about functions that influence consistency is a worthwhile area. Since the work discussed here began, several focused tools to study “OS jitter” have become available, such as PAL System Noise Activity Program (PSNAP) (PSNAP 2006). These tests are artificial benchmarks that focus on the consistency of OS performance within an SMP (node). It is not clear how OS jitter influences consistency of parallel applications, nor how to combine tools that measure SMP features with parallel applications.

9.7 Usability – Something for the Future

9.7.1 Summary

Scientists want to know how much harder it is to use HPC systems rather than their standard platform and tools. They want to know how much more effort is required to get a certain amount of work done on the HPC system rather than their desktops systems.

9.7.2 Impact

The impact of the usability work discussed here includes HPC sites adopting the NERSC User Survey methodology. It has also contributed to the successful evaluation of the CLE system, thereby accelerating its deployment world-wide.

9.7.3 Future Work

Usability is one of the most important aspects of system meeting their potential because, after all, is it the usage of the supercomputing systems that is really making a difference to the world. More work needs to done to provide quantitative methods for effectively surveying the users of the systems. Current research in usability is not well applied to HPC systems and provides an area for future work.

9.8 PERCU Summary

PERCU is a methodology that is very useful for sites when they are evaluating their high performance systems. There are several outcomes for this work in developing the various components of the methodology.

- Sites can use the PERCU method to develop a better set of measurable requirements for the purchase of their systems.
- Vendors can better respond to RFP's that were developed using the PERCU methodology because as more sites use it, hopefully vendors will adjust their processes and systems to be more efficient.
- As more sites use the software components of PERCU such as SSP, ESP, etc., they will have more accurate measurements of how the system is working and how effective it is for users. This will help the site in terms of future funding and measuring user satisfaction. Also, as more sites work with their vendors using the results of the benchmark testing, the vendors will be able to make their software better and

better. This will help them to have more information and succeed in being awarded more contracts for systems.

9.9 Chapter Conclusion

I want to thank everyone in the world of HPC; universities, scientists, vendors, users, government agencies, law makers, etc. for all the work in developing and using high performance computers. It is my sincere hope that the methodology of PERCU and its components will indeed help sites running large jobs to facilitate their users to solve some of the most complex and critical problems of the world and help make it a better place for all.

Appendix A. Additional Data

Additional data and information that was used in this work can be found at http://www.nersc.gov/~kramer/UCB/Dissertation/Data. There is also related links at http://www.nersc.gov/~kramer/UCB/Dissertation.

Appendix B. Characteristics Of The Systems

NERSC Systems Used in SSP Evaluations

System Name	Date of Acceptance Used	Processor Type and Speed	Interconnect	Total Number of CPUs	Total Number of Compute CPUs	SSP per CPU (MFlops/s)	System Wide SSP (GFlops/s)
NERSC 2 – Phase 1 – curie	September 1996	Cray T3E-600 – Alpha 5.2 @ 150 MHz	3D Torus	128	128	Using SSP-1 39 MFlops/s	Using SSP-1 3.9 GFlops/s
NERSC 2 – Phase 2 - mcurie	June 1997	Cray T3E-900 – Alpha 5.2 @ 225 MHz	3D Torus	512	512	Using SSP-1 46 MFlops/s	29.4 GFlops/s Using SSP-1 9.9.1
NERSC-3 – Phase 1 - gseaborg	October 1998	IBM Power 3 (Whitehawk II) –@ 200 MHz – 0.8 GFlops/s	IBM SP Switch	736	512	Using SSP-1 67 MFlops/s	Using SSP-1 34.8 GFlops/s required
Phase 2a - Seaborg	July 1999	IBM Power 3+ (Nighthawk) @ 375 Mhz – 1.5 GFlops/s	IBM Single-Single Colony Switch	3,328	3,136	Using SSP-1 114 MFlops/s	Using SSP-1 238.4 GFlops/s measured
Phase 2b - Seaborg	November 2000	IBM Power 3+ (Nighthawk) @ 375 Mhz – 1.5 GFlops/s	IBM Single-Single Colony Switch	3,328	3,136	Using SSP-1 116.6 MFlops/s	Using SSP-1 365 GFlops/s
NERSC-3E - Seaborg	December 2002	IBM Power 3+ (Nighthawk) @ 375 Mhz – 1.5 GFlops/s	IBM Double-Single Colony Switch	6,656	6,080	Using SSP-2 214.8 MFlops/s	Using SSP-2 1,305 GFlops/s Using SSP-4 890 Glop/s

NCSa - Jacquard	July 2005	AMD Opteron 2.4 GHz – 4.8 GFlops/s	Infiniband 12x backbone with IB 4x connection for each node	708	640	Using SSP-3 636.7 MFlops/s	Using SSP-3 413.9 GFlops/s
NCSb – Bassi	January 2006	IBM Power 5 @ 1.9 GHz – 7.6 GFlops/s	IBM HPS (Federation) with two planes per node	967	888	Using SSP-3 961 MFlops/s	Using SSP-3 923.3 GFlops/s
NERSC-5-DC	November 2007	AMD Dual Core @ 2.6 GHz – 5.2 GFlops/s	Cray Seastar 2.1 3-D Torus	19,488	19,320	Using SSP-4 994 MFlops/s Using SSP-5 699 MFlops/s	Using SSP-4 19.2 TFlops/s Using SSP-5 13.5 TFlops/s
NERSC-5-QC	September 2008	AMD Dual Core @ 2.3 GHz – 9.2 GFlops/s	Cray Seastar 2.1 3-D Torus	38,736	38,640	Using SSP-4 982 MFlops/s Using SSP-4 ~673 MFlops/s	Using SSP-4 37.98 TFlops/s Using SSP-5 ~26 TFlops/s

Table B-1: NERSC systems during the time of the SSP

Systems Used in Consistency Investigations

Category	Units	Cray T3ENERSC "mcurie"	IBM SPNERSC "Seaborg"	Compaq SC PSC "lemieux"	IBM Netfinity LBNL "alvarez"
Year of Installation		1997	2000	1998	2001
CPUs		Alpha EV57 based	Power 3+	Alpha EV 68	Intel Pentium III
Clock	MHz	450	375	1,000	866
Operations per Clock		2	4	2	1
MFlops/s per CPU	MFlops/s	900	1,500	2,000	866
CPUs per node		1	16	4	2
Memory per CPU	MB	256	1,000 to 4,000 most runs on 1,000	1,000	512
Caches		L1 – 8 KB L2 – 96 KB	L1 – Data 64 KB L1 – Inst 32 KB L2 – 8MB	L1 – Data 64 KB L1 – Inst 64 KB	L1 – Data 16 KB L2 – Inst 16 KB L2 – 256 KB
Memory Bandwidth	GB/s	.96 GB/s per CPU	1.6 GB/s per CPU Switched Base	8 GB/s per node 8 GB/s per node (2 GB/s per CPU) Switch based	.532 GB/s per CPU Shared Bus
Switch Technology		Custom	IBM "Colony"	Quadrics	Myrinet 2000
Switch Topology		3D Torus – Static Routing	Omega Network	Fat Tree	Fat Tree
Adapters per node		1	2 (2 is the default)	2 (1 is default)	1
Interconnect Bandwidth per adapter	MB/s	300	1,000	280	240
Latency (MPI)	□ sec	4.26	18.3	~5	11.8
Ping-Pong Test - MPI to MPI 1 task per node	MB/s	303	365	280 per rail	150
Number of CPUs		696	3,328 2944 compute (184 nodes)	3,000	170
Number of CPUs per node		1	16	4	2
Compilers		Cray	IBM	Compaq	PGH
O/S		Chorus kernel on compute nodes Unicos/mk on OS and Command nodes	Full Unix based AIX with SP software on all nodes	Tru 64 Unix	Full Red Hat Linux
Load		Heavy - >	Heavy - > 90%	Heavy – 75-	Very Light –

Category	Units	Cray T3ENERSC "mcurie"	IBM SPNERSC "Seaborg"	Compaq SC PSC "lemieux"	IBM Netfinity LBNL "alvarez"
		90%		85%	only user most of the time
Peak Aggregate Performance	TFlops/s	.63	5.0	6.0	.15
Latest LINPACK performance results	TFlops/s	.48	3.05	4.06	..09
Memory Ratio	B/Flop/s	.28	.67 (1.3 if a 32 GB node was used)	.5	.6
Communication Ratio (based on default number of adapters)	B/F	.333	.083	.0425	.138
Aggregate Main Memory Bandwidth for 128 CPUs	GB/s	122	205	256	68

Table B-2: Systems used for consistency investigation

Appendix C. Application Codes Used In the SSP Metrics

Application	Science Area	Basic Algorithm	Language	Library Use	SSP Version	System	Required Concurrency
NPB – MG Version 2.3 Class D	Various	Multi-Grid	Fortran	None	SSP-1	NERSC-3-	256
NPB – CG Version 2.3 Class D	Various	Conjugant Gradient		None	SSP-1	NERSC-3-	256
NPB – FT Version 2.3 Class D	Various	Fourier Transform	Fortran	None	SSP-1	NERSC-3-	256
NPB – LU Version 2.3 Class D	Various	LU Decomposition	Fortran	None	256		256
NPB – SP Version 2.3 Class D	Various	Pentadiagonal solver	Fortran	None	SSP-1	NERSC-3-	256
NPB – BT Version 2.3 Class D	Various	Block Tri-diagonal	Fortran	None	SSP-1	NERSC-3-	256
QCD	Quantum Chromo-dynamics	Conjugate gradient	Fortran 90	netCDF	SSP-1	NERSC-3-	Vendor selected concurrency
Camille	Climate	CFD, FFT	Fortran 90		SSP-1	NERSC-3-	Vendor selected concurrency
NWChem	Chemistry	DFT	Fortran 90	DDI, BLAS	SSP-1	NERSC-3-	Vendor selected concurrency
					SSP-2	NERSC-4/3E	256
SuperLU	General	Sparse Matrix	Fortran 77		SSP-1	NERSC-3	Vendor selected concurrency
						NCSa	32
tble	Materials		Fortran		SSP-1	NERSC-3	Vendor selected concurrency
CAM	Climate Navier Stokes CFD	CFD, FFT	Fortran 90			NCSa	32
					SSP-3	NCS-b	64
fvCAM	Climate Navier Stokes CFD – Finite Volume	CFD, FFT	Fortran 90	netCDF	SSP-4	NERSC-5	56 and 240
					SSP-5	NERSC-6	56 and 240 with strong scaling D Grid (~.5° resolution)

Application	Science Area	Basic Algorithm	Language	Library Use	SSP Version	System	Required Concurrency
							240 time steps
GAMESS	Chemistry	DFT	Fortran 90	DDI, BLAS	SSP-4	NERSC-5	64 and 384
					SSP-5	NERSC-6	384 and 1024 DFT gradient, MP2 gradient
GTC	Fusion	Particle-in-cell and Finite Difference	Fortran 90	FFT(opt)	SSP-2	NERSC-4/3E	256
					SSP-3	NCSb	64
					SSP-4	NERSC-5	64 and 384
					SSP-5	NERSC-6	384 and 1024 DFT gradient, MP2 gradient
MADcap	Astrophysics	Out of core Power Spectrum Estimation	C	Scalapack and large scale I/O k	SSP-2	NERSC-4/3E	484
SEAM	Climate	spatially-decomposed finite element			SSP-2	NERSC-4/3E	1,024
NAMD	Chemistry	Molecular dynamics			SSP-2	NERSC-4/3E	1024
						NCSa	32
Chombo	CFD (Combustion, Fusion, Climate)	Adaptive Mesh Refinement	C++			NCSa	32
MADbench – a special benchmark version of MADCAP	Astrophysics (HEP & NP)	Out of Core Power Spectrum Estimation	C	Scalapack and large scale I/O	SSP-4	NERSC-5	64, 256 and 1024
MILC	QCD (NP)	Conjugate gradient sparse matrix; FFT	C and Assembler	none	SSP-2	NERSC-4/3E	512
					SSP-4	NERSC-5	64, 256 and 2,048
					SSP-5	NERSC-6	256, 1024 and 8192 Weak Scaling 8x8x8x9 Local Grid - ~70,000 iterations
Paratec	Materials (BES) Nanoscience	3D FFT, DFT, BLAS3	Fortran 90	Scalapack, FFTW	SSP-1	NERSC-3	Vendor Selected concurrency
					SSP-2	NERSC-4/3E	128
					SSP-3	NCSb	64
					SSP-4	NERSC-5	64 and 256
					SSP-5	NERSC-6	256 and 1024 Strong scaling

Application	Science Area	Basic Algorithm	Language	Library Use	SSP Version	System	Required Concurrency
							686 Atoms, 1372 Bands, 20 iterations
PMEMD	Life Science (BER)	Particle Mesh Ewald	Fortran 90	none	SSP-4	NERSC-5	64 and 256
IMPACT-T	Accelerator Physics	PIC, FFT component	Fortran 90		SSP-5	NERSC-6	256 and 1024 strong scaling 50 particles per cell
MAESTRO	Astrophysics	Low Mach Hydro; block structured-grid multi-physics	Fortran 90	Boxlib	SSP-5	NERSC-6	512 and 2048 weak scaling 16x 32^3 boxes per processor; 10 timesteps

Table C-1: A summary of all the application benchmarks used at NERSC at different points in time.

Appendix D. NAS Parallel Benchmarks Used for Consistency Testing

LU – The LU (Lower Upper) benchmark solves a finite difference discretization of the 3-D compressible Navier-Stokes equations through a block lower-upper-triangular approximate factorization of the original difference scheme. The LU factored form is solved by symmetric successive over relaxation (SSOR) that solves a 5 by 5 blocked regular sparse matrix. The code requires a power-of-two number of processors and is load-balanced. The Computation Intensity is high and LU sends many small messages.

FT – A 3-D Fast Fourier Transform (FFT) Partial Differential Equation with a 3-D array of data is distributed according to the z-planes of the array. One or more planes are stored in each processor. The forward 3-D FFT is performed as multiple 1-D FFTs in each dimension, first in the x- and y- dimensions. This can be done entirely within a single processor with no inter-processor communication. An array transposition is performed, which requires an all-to-all message exchange. Thus FT shows big, bursty communication patterns among all nodes in between periods where all nodes are computing.

EP - (Embarrassingly Parallel). Each processor independently generates pseudorandom numbers in batches and uses these to compute and tally pairs of normally-distributed numbers. No communication is needed until the very end, when the tallies of all processors are combined. This test was included to give a baseline for CPU performance.

Appendix E. SSP Related to Productivity

SSP is an extension of the productivity based method described in “A Framework for Measuring Supercomputer Productivity” (Snir and Bader 2004). In particular, SSP relates to the (Snir and Bader 2004) dual problem and addressing the issue of multiple problems in a workload, by incorporating multiple applications and problem sets. However, SSP is much closer to actual assessments methods used in practice.

As a high level summary, (Snir and Bader 2004) defines *productivity*, Ψ, for a system, S, based on a problem, P, that has a time-to-solution of T. The Utility, U(P,T), of a system is based on the time to solve the problem. There are several types of utility in the paper, including deadline based, decreasing value and constant. Deadline based assigns the full utility value a task to complete before a deadline and zero value after the deadline. Constant utility means there is no change in the utility value over time, so it does not matter how long it takes to do a task. Decreasing utility assign some function to utility that decreases at a prescribed rate (linear, exponential, etc.) over time. Cost, C(P,S,T), depends on P, T and S. Productivity is Utility divided by cost.

$$\Psi\left(P,S,T,U\left(P,T\right)\right)=\frac{U(P,T)}{C(P,S,T)}$$

(Snir and Bader 2004) state the goal of a HPC system is to "minimize the cost of solving P on system S in time T." Below is a discussion of the similarity, overlap and differences between the SSP method and the concepts outlined by the paper.

The similarities between the (Snir and Bader 2004) and SSP approaches are that both propose methods to assess how well computer systems meet the needs of users to solve problems by providing a single composite metric. Both deal with estimates of the overall work a system is capable of and the cost of the system in making value judgments as to which system would be best to deploy. S&B's Utility is similar to SSP's Potency. In the case where SSP is defined by a single problem and a single data set (where I=1 and J=1), the two methods are the identical. Likewise, for a single problem and data set, productivity is synonymous with SSP value.

Both methods include the ability to incorporate multiple cost components, both one-time costs and on-going costs. The discussion in (Snir and Bader 2004) concentrates on system costs and costs to port/optimize code P to a machine S. The SSP method includes total

cost of ownership (HW, SW, maintenance, power, space, etc.). SSP can also include porting costs, but so far it has not been used for that purpose. Finally, (Snir and Bader 2004) discusses the concept of productivity and application metrics as a means of expanding their approach to include more than one application with one data set. This is comparable to the entire PERCU Method (the overall method of evaluating systems based on Performance, Effectiveness, Consistency, Reliability and Usability).

There are differences between the two approaches as well. Specifically, (Snir and Bader 2004) has the goal of minimizing the cost for solving a problem. However, more often than not, there is a set cost to spend on a system and multiple things for the system to do. To address this aspect, (Snir and Bader 2004) B introduces the “Dual” problem of trying to optimize the system to produce the best time-to-solution it can for a fixed cost. This is similar to the goals of SSP, but the full goal of a SSP evaluation is to get the best system to be used on a range of problems. In SSP, while cost can be optimized, it is used as the normalizing factor across alternative systems rather than the objective function for minimization.

(Snir and Bader 2004) discusses optimizing the solution of one program. SSP focuses on multiple applications and data combinations. The (Snir and Bader 2004) approach to multiple applications uses the modeling of the applications in a workload and using the models as productivity metrics. This approach requires valid statistical sampling methods of the target workload, essentially a Workload Characterization Analysis (WCA). (Mashey, 2004) (see Section 3.11) points out that being able to do an evaluation for a large workload that is statistically "pure" is very hard and takes a lot of effort, so most times it is not done, only approximated. Using Mashey's descriptions, the S&B method to evaluate multiple codes and data corresponds to a WAC with a uniform workload while SSP is a SERPOP analysis (discussed in Section 3.13 above).

(Snir and Bader 2004) does not address loss of opportunity as part of utility. The question "*What if the next science or engineering breakthrough waits for 20 years?*" is an important aspect of investing in the capability of computing facilities. One could theoretically adapt the S&B constant utility function so it has a probability of increasing significantly based on the probability function of having a breakthrough result occur at some point in time. This is not done in (Snir and Bader

2004). SSP implicitly provides this value as part of the system's potential.

(Snir and Bader 2004) was motivated by the DARPA High Productivity Computing System (HPCS) Program (HPCS n.d.) which is spending 7+ years to design a new, Petascale computing system based in part on productivity goals. Conversely, SSP is motivated by the need to evaluate the best system out of several which have been proposed for a workload that includes a variety of uses. Similarly, (Snir and Bader 2004) is designed to determine the cost to deploy a computational system to solve a single problem in an expected time. For the case of multiple problems running on the same system, (Snir and Bader 2004) mentions the future potential to allocate the costs to multiple problems.

In summary, SSP can be viewed as an extension of the ((Snir and Bader 2004) dual problem and addressing the issue of multiple problems in a workload, by incorporating multiple applications and problem sets. SSP is much closer to how most actual assessments are done. Indeed, SSP is used in practice to make evaluation and selection determinations

Appendix F. ESP-1 – The First Version

ESP-1 was the first implementation of the ESP concept. It was initial implementation developed specifically to reflect the NERSC workload. Any other site can adopt the steps described here using their own workload or they could use ESP-2, which is more portable and may be adjusted for different workload distributions.

A goal of the ESP-1 test was to represent the user workload, so the distribution of job sizes and runtimes in the ESP-1 suite were designed to roughly match the distribution of jobs running on NERSC production systems at the time. The types of applications were the same used for the SSP test and the concurrency was similar to the size of jobs running across the system. The one exception to matching the workload is that ESP-1 runtimes were scaled back so that the test took a practical elapsed time of approximately 2 to 4 hours.

ESP-1 used applications in the job mix that originated from user codes that were used in production computing at NERSC. While each facility will have its own workload distribution, the basic process described here can be applied to any workload. Furthermore, the job mix profile was designed to span the diverse scientific areas of research. Attention was paid to diversify computational characteristics

such as the amount of disk I/O and memory usage. Applications and problem sets were selected to satisfy the time and concurrency constraints. The number of instances (*Count*) of each application/problem was adjusted such that aggregate CPU-hours of the test reflected the workload profile. Table F-1 shows the job mix for the ESP-1 benchmark with the elapsed times for each job on the T3E and SP/P3.

Application	Discipline	Size	Count	T3E	SP
gfft	Large-FFT	512	2	30.5	255.6
md	Biology	8	4	1208.0	1144.9
md		24	3	602.7	583.3
nqclarge	Chemistry	8	2	8788.0	5274.9
nqclarge		16	5	5879.6	2870.8
paratec	Materials-Science	256	1	746.9	1371.0
qcdsmall	Nuclear-Physics	128	1	1155.0	503.3
qcdsmall		256	1	591.0	342.4
scf	Chemistry	32	7	3461.1	1136.2
scf		64	10	1751.9	646.4
scfdirect	Chemistry	64	7	5768.9	1811.7
scfdirect		81	2	4578.0	1589.1
superlu	Linear-Algebra	8	15	288.3	361.2
tlbebig	Fusion	16	2	2684.5	2058.8
tlbebig		32	6	1358.3	1027.0
tlbebig		49	5	912.9	729.4
tlbebig		64	8	685.8	568.7
tlbebig		128	1	35,0.0	350.7

Table F-1: ESP-1 Application job mix used at NERSC for the T3E and IBM SP/3 systems.

Before the ESP-1 test could run, each application was run on the target system for each concurrency and data set. This was typically done as part of other testing for performance. Once this was done, a “best possible” time, T-BEST was calculated using a bin packing

algorithm that minimized the elapsed time to run all the jobs, given a submission schedule for the jobs and the target system configuration. The “best possible” time is the equivalent of the theoretical minimum time to run the ESP-1 workload and it can be compared to the actual times.

The ESP-1 test ran roughly four hours on 512 CPUs of the NERSC T3E (*mcurie*) and approximately 2 hours on the IBM SP – Power 3+ (*seaborg*) system. While not perfect, the length of time is a practical compromise between a longer simulation that may more precisely represent actual usage and a shorter time that is more suitable to benchmarking that works on existing systems and does not adversely perturb real operations. Further, long running tests are difficult for vendors to perform since there is limited access to demonstration and testing systems.

Based on the experiences of ESP-1, a goal was set for, ESP-2 to be a proxy for the adverse impacted by the time and resources spent in system administration tasks, which are tasks that impact day to day usage. This secondary goal was dropped for the second generation of the ESP.

Full Configuration Jobs

Two full configuration jobs, called Z tests, are used to address the operational change paradigm. In this case, full configuration means a job that uses all the CPUs on the system. The first Z test is submitted after the job blocks previously discussed and mimics a “run me now” requirement that may be representative of deadline processing workload. It must be scheduled to run before any other work, but currently running jobs may, but do not have to, complete.

There are numerous ways a system can deal with this requirement based on the functionality of the system job schedule, including:

- Check pointing the running work, running the full configuration job, and then restoring the running workload
- Using preemptive scheduling that suspend the running workload and scheduled the Z test
- Not scheduling any more work and gradually idling the system until enough resource is available for the Z test
- Idling the system, but scheduling the jobs as backfill that can complete before the longest running job completes

- Killing some or all of the running jobs, running the Z test, and then restarting the jobs that were killed

The second full configuration Z test represents a job that uses all the system resources and needs to be completed within a reasonable amount of time – but not immediately. This would be the type of job that cannot be put off indefinitely but can wait for time period such as a job whose submitter needs the output the following day.

An example is a system whose level of service objective is to run very large jobs within a week or a several days. However, such jobs cannot be put off forever since in real life, there is always more work entering the system. Seldom are HPC systems underutilized so they run out of work and go idle. Because the ESP test is finite, it would be incorrect and trivial to let the second Z test wait until all other work completes. Hence, the second Z test has to complete before 90% of the rest of the workload has completed. That is, it cannot be the last job running. The second Z test gives the scheduler more options to use, but still requires an operational shift in order to complete successfully within the limits of the test. Few schedulers, however, have the ability to specify this “complete before this time” provision or

lack features that allow for deadline processing. In such cases, the second Z test is also specified as “run now”.

Data from the ESP-1 Test Runs

Two test runs were completed on the T3E* and one run on the IBM SP. In both T3E cases, a separate queue was created for full configuration jobs. The full configuration jobs could thus be launched immediately on submission, independent of the queue of general jobs. Process migration/compaction was also enabled for both runs. In the first run, labeled *Swap*, the system was oversubscribed by a factor of two and the jobs were gang-scheduled with a time slice of 20 minutes using standard system software. A single NQS (Network Queuing System) queue was used for the general job mix. In the second run, labeled *NoSwa*p, the system was not oversubscribed. Each job ran uninterrupted until completion. Six queues for different maximum partition sizes; 256, 128, 64, 32, 16, with decreasing priority were used. For each run, the time on the X axis was normalized to the time best case time.

* The runs were done the NERSC T3E-900 – named *mcurie*, with 696 900 Mhz Alpha EV56 CPUs. Each CPU has 256 MB of memory and the system has a total of 2.5 Terabytes of local disk.

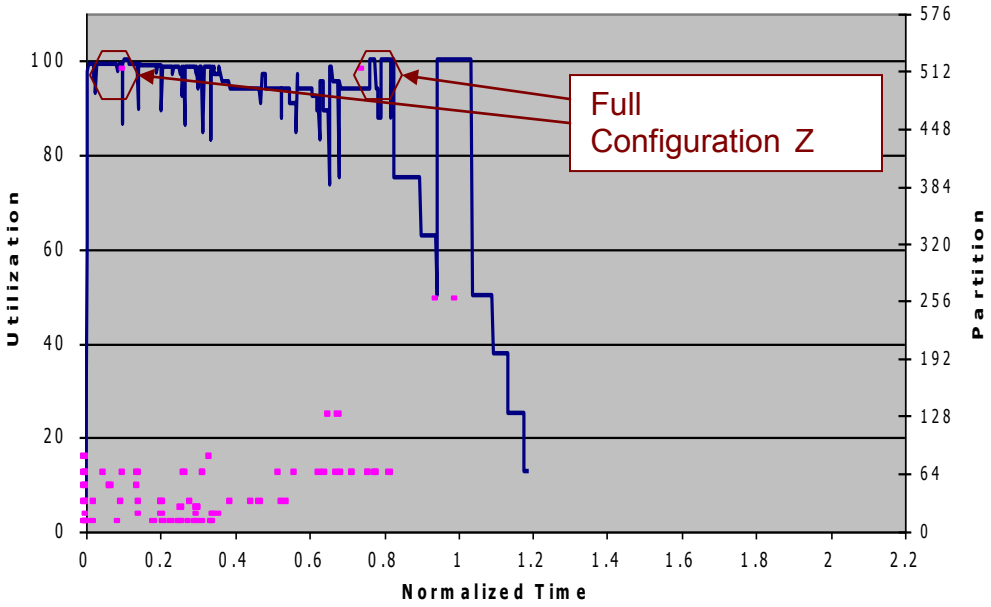


Figure F-1: T3E Job Chronology with the Swap run. The magenta marks show when particular jobs begin execution. The line shows the instantaneous utilization of the system. The time access is normalized to the best time. Note the two Z tests run at time 0.15 and 0.8. This shows that the ESP-1 took about 20% longer to complete then the best bin packing would predict.

Figures F-1, F-2, F-3 and F-4 show the details of the runs where the instantaneous utilization (line) is plotted against time and the time axis has been rescaled by the theoretical minimum time, T-BEST. Additionally, the start time for each job is indicated by an impulse marker (magenta) where the height equals the size, in processors, of that job. It is possible to see the start times for the two full configuration jobs since they are at the top of the chart.

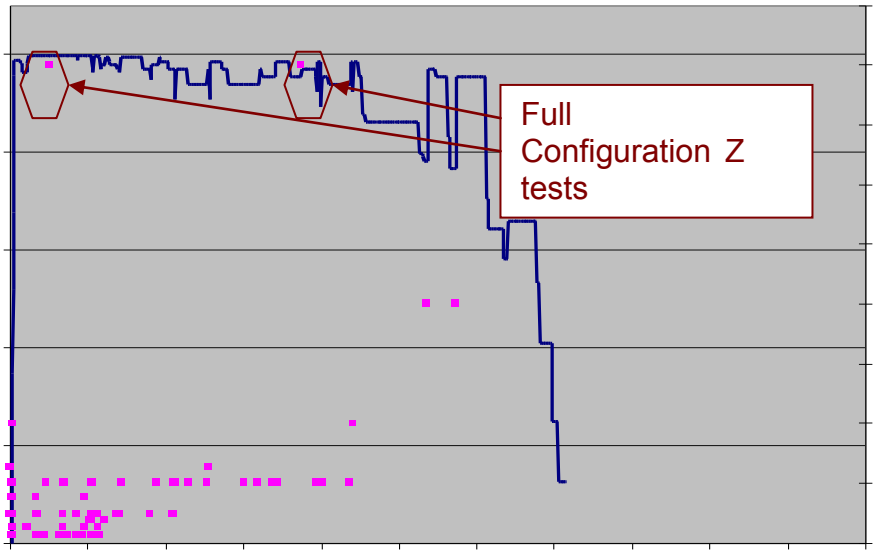


Figure F-2: The T3E Job Chronology for the NoSwap run. The time for this run to complete was about 1.4 times the bin packing prediction.

A different job scheduler, IBM's *Loadleveler*, with different management features existed on the IBM SP*. To encourage the best possible scheduling, two classes (queues) were created in *Loadleveller*; a general class for all jobs and a special high priority class for the full configuration jobs. At the time of the test, it was not possible to *selectively* backfill with *Loadleveller*. Runs shown in Figure F-3 indicate that backfill would defer launching of the full configuration Z job until the end of the test. This clearly violated the intent of the test to represent the real world since new jobs would always flow into the

* The SP system is 604 nodes of 2 CPU SMPs. The CPUs are "Winterhawk 1" CPUs – which is a Power3 PCPU running at 200 MHz. Each node has 1 GB of memory and is connect with IBM's TBMX-3 switch.

systems, meaning the Z job would never run. To address the limitation in *Loadleveler*, backfill was implicitly disabled by assigning large wall clock times (several times greater than the elapsed time for the complete test) to all jobs. Thus *Loadleveller* was reduced to a strictly FCFS (First Come First Served) strategy. The resulting run is shown in Figure X-Y. It is interesting to note that the T3E and the IBM SP Power 3 systems are approximately the same peak computational power.

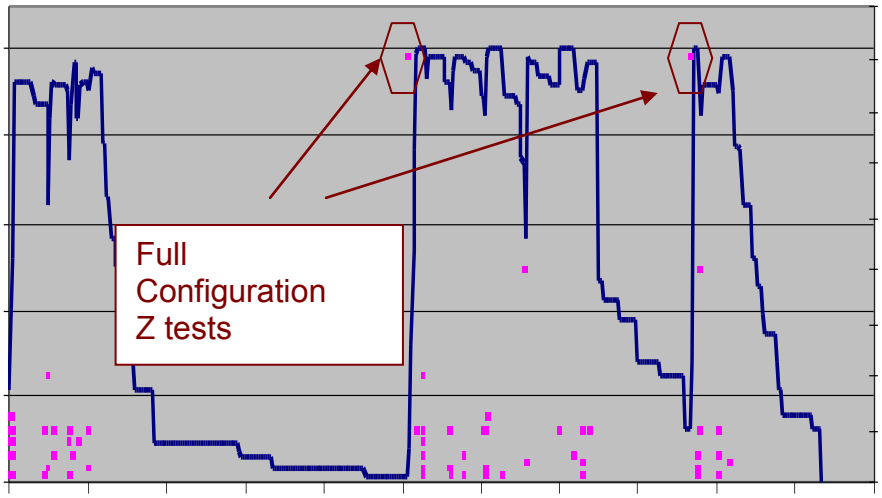


Figure F-3: The IBM SP Job Chronology for the Swap run. Note the dramatic drop in utilization as the system accumulates CPUs to run the full configuration Z jobs. The test ran more the twice as long as the expected best time.

On submission of the full configuration jobs, a considerable amount of time was spent waiting for running jobs to complete. This is evident in Figure F-4, which shows two large regions where the instantaneous utilization drops to a very low value. The time lag to run preferential jobs is indicative of the difficulty in changing modes of operation on the IBM SP for that version of the system. This is important for sites that routinely change system characteristics, for example between interactive and batch or between small and large partitions. The best remedy would be to either checkpoint or dynamically swap out running jobs.

As seen in Figure F-4, the Best Fit First (BFF) mechanism on the T3E deferred larger scale jobs (128 or greater MPI tasks) until the end. Consequently, at the end of the test there were large gaps that could not be filled by small jobs. On the IBM SP, a First Come – First Served (FCFS) strategy was indirectly enforced, which can be seen illustrated in Figure F-1 where the distribution of job start times is unrelated to job size. It is evident from Figures F-1 and F-2, that a significant loss of efficiency on the T3E was incurred at the tail end of the test. In an operational setting, however, there are usually more jobs to launch that have entered the system. Hence, the value of having a quantitative test

out-weighs the fact it being finite does not completely represent real operating conditions.

The distribution of start times is qualitatively similar between the *Swap* and *NoSwap* runs on the T3E, although the queue was set up differently. In the second test run, increasingly higher priorities were deliberately assigned to larger partition queues in an attempt to mitigate starvation. However, shortly after the start of the test, it was unlikely that a large pool of idle processors would become coincidently available. In this scenario, the pattern of job submission reverted back to BFF and the queue set up had little impact. On the other hand, there was considerable difference in efficiency between the two T3E runs. This was attributed to the overhead of swapping, which was significant when the oversubscribed processes could not simultaneously fit in memory resulting in significant I/O to swap the processes between memory and disk.

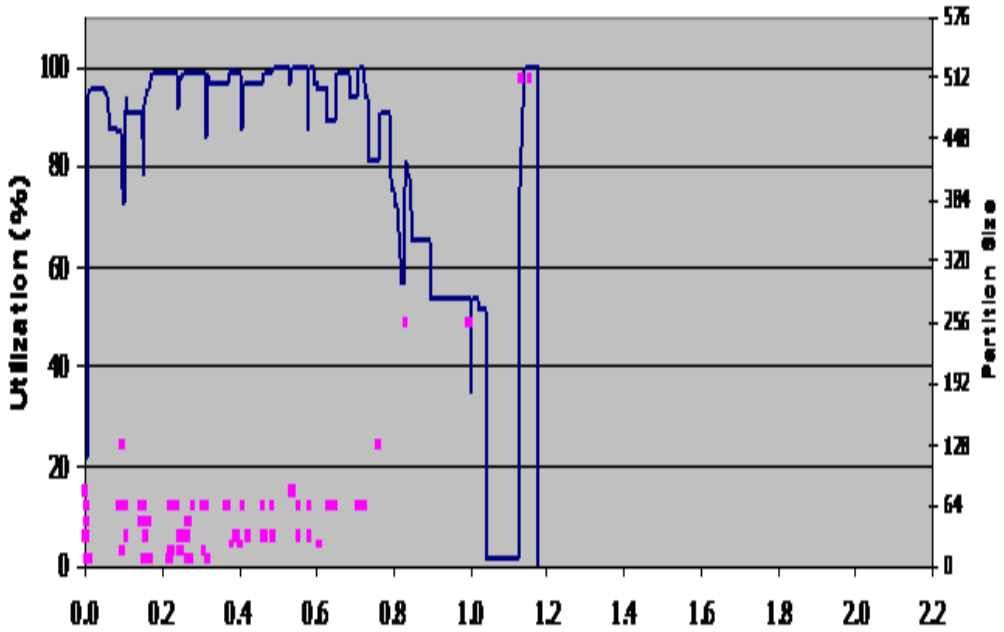


Figure F-4: SP Job Chronology with backfill decreases the runtime of the ESP-1 test, but the fact the two full configuraiton jobs are pushed to the end of the test clearly violates the conditions of the test.

ESP-1 Summary of Results

The results of the ESP-1 test for the T3E and the SP are summarized in Table F-2.

	T3E Swap	T3E NoSwap	SP
Available processors	512	512	512
Job mix work (CPU-sec.)	7437860	7437860	3715861
Elapsed Time (sec.)	20736	17327	14999
Shutdown/reboot (sec.)	2100	2100	5400
E - Efficiency (w/o reboot)	**70%**	**84%**	**48%**

Table F-2: ESP-1 Results on the T3E and the SP shows that scheduling systems at on the T3E could accommodate operational paradigm shifts better than the IBM SP.

These results show that the T3E has significantly higher utilization efficiency than the SP with the same operational paradigm and job mix. This difference is mainly due to the lack of an effective mechanism to change operational modes in a reasonably short time period, such as is necessary to immediately launch full configuration jobs.

Validation of ESP-1

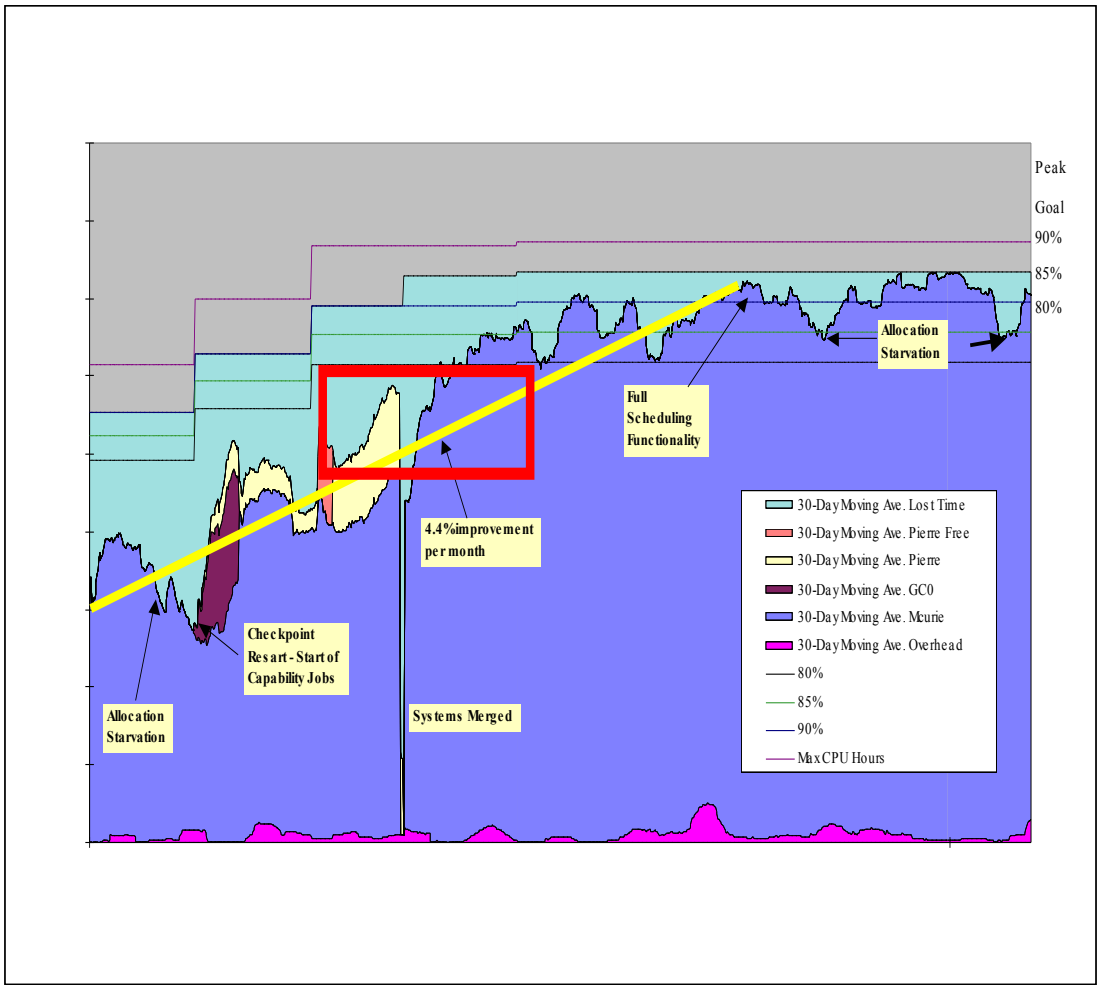


Figure 9-1: Actual Utilization of the NERSC T3E over a 3 year period. The date with the blue color is the 30 day moving average of the CPU used by user applications. The T3E usage increased with the introduction and improvement of system software.

It is important to validate the results of any benchmark test with real data. It is now possible to do so by reviewing the utilization data on the T3E and the SP for the actual workload the ESP test was first designed to mimic. Figure F-5 shows the T3E utilization for a period of 3 years. Figure F-6 shows the SP utilization for 8 months that system was in service before being upgraded.

The utilization of the T3E in the first 30 days was 57%. Utilization was limited by the fact the T3E needed to assign jobs CPUs in a contiguous block based on logical node numbers which were assigned to physical nodes at boot time. Contiguous logical nodes simplified the internal interconnect routing. This may improve communications for some applications, but it means there is the fragmentation of unused processors that are left idle since there are no jobs of the size that can run. Indeed, is it possible that seven long running small jobs (4 processors), poorly positioned, could prevent any jobs 64 or more CPUs from starting, even though the system had 512 CPUs. It is important to note that during this period, in order to develop and test of the more advanced scheduler functions, (Blakeborough and Welcome 1999) the T3E was taken out of service and rebooted at least two times a week – for 6 hours each time. The introduction of the job migration facility, shown on the Figure F-5, followed by the deployment of checkpoint restart, allowed increasingly effective system operation while running full configuration jobs every night. This period, shown in the red box on Figure F-5, has the same level of function used for the ESP-1 test runs. It shows the actual utilization is roughly 70% utilization, essentially the same as the ESP-1 test rating for the T3E Swap case in Table D-5.

The utilization on the SP started out higher than on the T3E since the backfill function was immediately available and jobs could be assigned to any set of nodes to a job. With backfill on, the usage is well over the expected utilization shown by the Efficiency Rating indicated by the ESP without backfill. However, the system was not able to respond well to full or close to full size jobs. Manual intervention was required to start each large job. Since the T3E could run large jobs well and both systems were in service and available to all users, very little of the work was done by jobs with less than half the maximum number of CPU's.

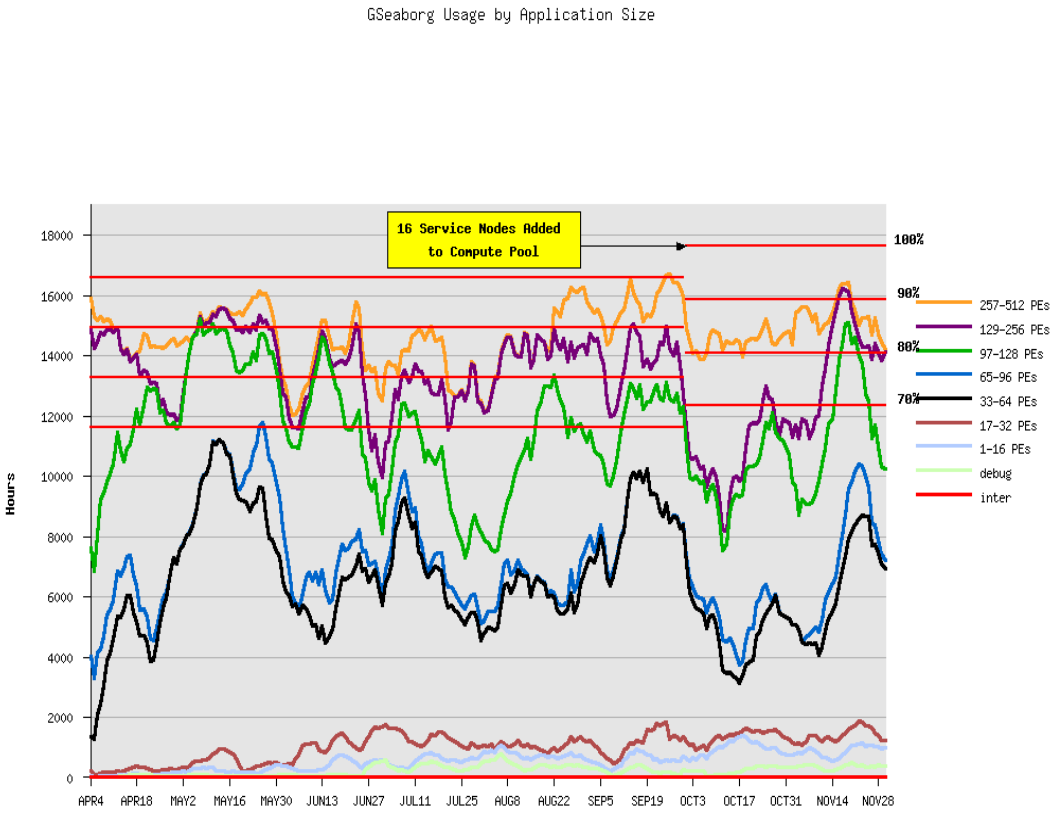


Figure F-9-2: SP Workload – Cumulative CPU Time by Job Size on the SP for an 8 month period.

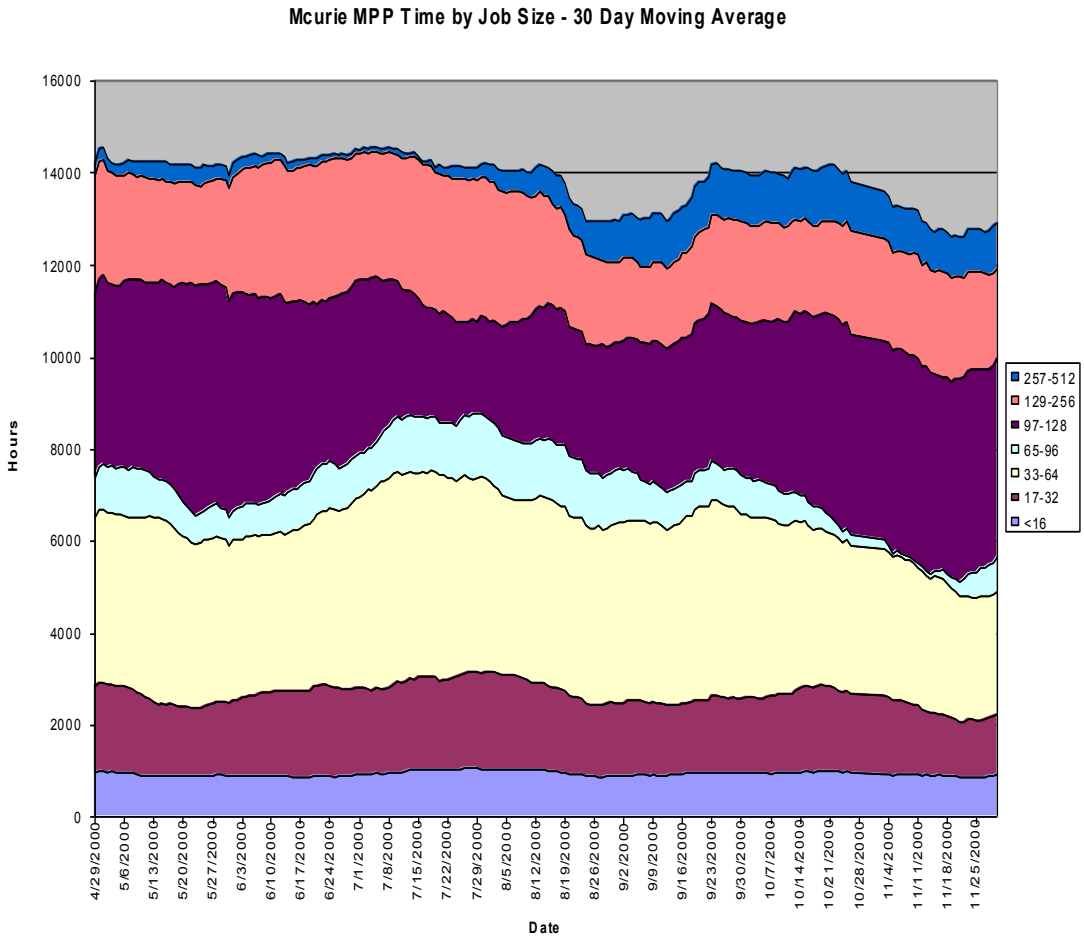


Figure F-9-3: T3E Workload – Cumulative CPU Time by Job Size for an 8 month period. The total number of computational CPUs is 696.

Figures F-6 and F-7 show that the actual workloads on the NERSC T3E and the SP during the same time. The workloads have several common characteristics, which is not surprising, given they are similar in size and capability, and support the same user community. The first observation is that each system runs a number of jobs that are near full

configuration. In Figure F-6, for the last two months of the period, 10% of the CPU time on the IBM SP was used by full configuration (512 CPU) jobs. A similar result is shown in Figure F-7 for the T3E. The other observation is that the vast majority of the CPU time on both systems goes to jobs of substantial size. In the T3E case, more than 50% of the computational time was used by jobs using 128 CPUs or more – which is 1/4 of the system.

Even with backfill on in the production system, Figure F-6 shows that eventually large jobs do run on the IBM SP and make up a significant part of the SP workload. What is not shown in a graph is that the length of time large jobs wait for service is much longer on the SP, because the system has to age large jobs a very long time to get processors assigned, or alternatively, manual intervention is used.

Table F-3 compares ESP-1 tests on the SP runs with and without backfill. As stated above, backfill violates the test rules because the full configuration jobs are not processed in a timely manner. Nonetheless, as a validation data point, the ESP-1's effectiveness rating is an indicator of the utilization a system is able to support, we look at what the ratings would be with backfill, as seen in E = 84%. This is very

close to the observed utilization while running under the exact same operational parameters on the IBM SP as shown in Figure F-6.

	SP without backfill	SP with backfill but violates test parameters
Available processors	512	512
Job mix work (CPU-sec.)	3715861	3715861
Elapsed Time (sec.)	14999	8633
E - Efficiency (w/o reboot)	**48%**	**84%**

Table D-3: ESP-1 Results for two scheduling methods on the IBM SP/3

Limitations of ESP-1

The ESP-1 test, while effective at NERSC for the T3E and the IBM SP, had several limitations that prevented it from becoming a more general test. These limitations included the fact ESP-1 was based on a snapshot in time of the NERSC applications and workload. ESP-1 proved labor intensive to scale to other system sizes. For example, when NERSC upgraded the IBM SP* to an IBM SP/Power 3+, the application sizes and number of codes had to be completely redone. In order to address these limitations and to make ESP more acceptable for others for use, the ESP-2 test was created.

* The upgrade replaced the gseaborg system with seaborg – a 3,328 process Power 3+ system with 16 processors per node, each running at 375 MHz. Each node had a dual plane High Performance (Colony) Switch.

Appendix G. Assessing Batch Schedulers with ESP-2

In this section, an example of how ESP-2 was used to assess a variety of batch scheduling software utilities for cluster computer systems is discussed. This study was initiated in 2004-2005 to evaluate the features and performance characteristics of the then currently available Linux based, open source and commercial schedulers, in anticipation of additional commodity based clusters becoming available.

The ESP-2 methodology was useful in evaluating software releases. In addition to distinguishing characteristics of the different schedulers, several of the evaluated job scheduling systems made improvements to the features and performance of their systems in order to improve test results, which for the most part, became permanent product features in difference resource managers. In other words, ESP-2 was able to contribute to the improvements in several available resource managers, which in turn contributed to other organizations using these packages.

Taking a larger perspective with ESP-2 on available scheduling systems that could be integrated into environments such as NERSC

was possible because of the ESP-2 test. The assessment investigated the systems features to schedule work for robustness, scalability, performance, features, and effectiveness.

Layered Functions Of The System

Large Linux and Unix clusters offer a range of software for resource management features at different layers of the system. Sometimes one software package combines functions, but in general the layered functions can be thought, from high level to low level, of as follows.

A job management – often call batch – system that provides users and system managers the ability to submit work packages (jobs) to be run by the system. Examples include NQS, PBS and Condor.

A job scheduler is used to select and prioritize which jobs should run before others. While all batch systems have a built in scheduler, many provide an external API (Application Program Interface) to allow more sophisticated and/or site specific scheduling to be performed. An example is the Maui scheduler.

A resource scheduler is a system function that applies to batch and non-batch (interactive) activities which finds available resources and

accumulates sufficient resources, typically nodes, to start processes and jobs.

A process scheduler, using fine grained kernel scheduling within a single operating system image.

Table G-1 lists some of the features of the scheduling systems tested with ESP-2. While ESP-2 was the major filter by which we considered whether to continue with software from this list, it was not efficient to set up and run ESP-2 with schedulers which obviously did not meet the needs and feature requirements of NERSC.

A few particularly important features were chosen that were considered necessary for a batch scheduler to be considered. These features were: Linux support, low cost, open source, parallel job support, prologue and epilogue capability, and the ability to use third-party schedulers (particularly the Maui Scheduler). Each scheduling package was installed and tested on the same hardware configuration.

The batch schedulers considered were identified through communal knowledge of what was available (and desirable) when the investigation began in mid 2004 and included The University of Wisconsin's Condor (Condor n.d.), the LLNL Linux Project's SLURM

(Simple Linux Utility for Resource Management) (SLRUM n.d.), the Maui High Performance Computing Center and University of New Mexico's MauiME (Maui Molokini Edition) (Maui n.d.), Sun's SGE (Sun Grid Engine) (SGE n.d.), Platform Computing's Load Sharing Facility (LSF), Fermi National Accelerator Laboratory's FBSNG (Farms Batch System Next Generation) (FBSNG n.d.), OpenPBS (PBS n.d.), and PBSPro (Jones 2004). Systems that ran on proprietary hardware were not considered.

Condor, at the time, lacked a number of the base features needed and also support for parallel MPI jobs. Later, parallel support was included in Condor, but only for MPI-based environments for which opportunistic scheduling was used. SLURM, an alternative to PBS, lacked support for Myrinet and Maui Scheduler. Support for Maui was in development; however, the SLURM team had no plans for Myrinet compatibility, since its creators (LLNL Linux Project) only used a Quadrics interconnect. Since the test platform was Myrinet based, the lack of support eliminated SLURM from further consideration. Similarly, IBM's Loadleveler scheduler, discussed separately5.7 above, was not evaluated because it was not available on the hardware used for the evaluation.

FSBNG lacked prologue and epilogue support, immediately disqualifying it, since such a capability was essential to setting up the secure environment on Lawrence Berkeley National Laboratory's (LBNL) Alvarez system in which multiple users run jobs. Since this could not be accomplished at the user level, it had to be done by the scheduler. PBS-Pro, was a commercial product, had basically the same functionality as OpenPBS, so OpenPBS was evaluated during this investigation.

Hence MauiME, Sun Grid Engine (SGE), LSF, and OpenPBS (Table G-1) were included in the ESP-2 evaluation. While LSF did not meet the low-cost criterion, it was included as a candidate because it, along with OpenPBS, was already used within NERSC, and both could serve as benchmarks for comparison.

Feature	OpenPBS	LSF	MauiME	SGE
1.1 Backfill	X	X	X	-
1.2 Backfill “end time” designation	X	X	-	-
1.3 Preemption	X	X	forthcoming	-
1.4 Gang Scheduling	-	-	-	-
1.5 Sessions	-	-	-	-
1.6 Scheduler has ultimate control of processes	-	X	-	?
1.7 Migrate Jobs and Processes	-	X	-	X
1.8 Advanced Reservation	X	X	X	-
1.9 Resource Dedication	X	X	X	-
1.10 Queue complexes	X	X	-	X
1.11 Control queues on a per node basis	-	-	-	?
1.12 Suspend and resume jobs	X	X	-	X
1.13 Configurable Resource Definitions	X	X	X	X
1.14 Parallel job support	X	X	X	X
1.15 Pluggable Scheduler	X	-	X	-
1.16 Linux and Sun support	X	X	X	X
1.17 Checkpoint/Restart	On SGI, Cray only	X	X	X
2.1 Ability to force a job to run outside of prioritization	X	X	X	X
2.2 “No-preemption” marking for jobs and queues	-	-	-	-
2.3 Ability to define and enforce limits	X	X	X	X
2.4 Highly detailed logs	-	X	X	?
2.5 Set queues to be empty at a certain time	-	X/-	X	-
2.6 Extensive API	-	X	X	?
2.7 Override system configuration with node-specific configuration	-	X	X	X
2.8 Custom job prioritization	X	X	X	?
2.9 Fair Share Scheduling	X	X	-	X
2.10 Open Source	-	-	X	X
2.11 Low cost	X	-	X	X
2.12 Robust and Fault Recoverable	-	X	X	?
3.1 Ability to run MPI jobs from the command line	X	X	-	?
3.2 Useful, detailed debugging info when a job fails	-	X	X	X
3.3 Prolog/Epilog system	X	X	X	X
3.4 Simple Scripting	-	-	X	X
3.5 Output immediate results to submission directory	-	-	X	X
3.6 Pre-exec conditions	X	X	X	With scripting?
3.7 User Space Checkpoint/Restart	-	X	-	X

Table G-1: Evaluation Features of the Evaluated Scheduling Software Systems as of June 2004

Installation and Basic Operation

MauiME, which has since evolved into the Torque scheduling system offered by Cluster Resources Inc., is a Java based complete scheduler and resource management system that comes out of the Maui Scheduler project. The developers have proven that it runs successfully not only on Linux and FreeBSD (FBSD n.d.), but also on 64-bit versions of Linux.

Being Java based, MauiME requires a Java Virtual Machine (JVM) to run. This is a benefit for heterogeneous environments, in terms of client deployment. MauiME does not have a way to submit a job with a given priority. A job must be submitted and then have its priority altered. Submitting jobs was more tedious than with most other batch schedulers tested. A job had to have a job description file as well as a separate batch script. If a sizable number of jobs were concurrently submitted (as ESP-2 does), the scheduler would only use a subset of those jobs to determine how to schedule the whole lot. In addition, there is no way to create node exclusivity—one processing element (PE) per node—so that a node is always the smallest resource unit.

SGE from Sun Microsystems is also a complete scheduler and resource manager. It is free and fully functional, with copious documentation. An attribute of SGE is its versatile graphical user interface (GUI), something all the other candidates lacked. With the GUI you can see the status of hosts, queues, jobs, and processes involved in the execution environment and do virtually all of the administrative tasks that can be done with command line utilities.

A less attractive attribute of SGE, was that the scheduler applied priorities in terms of job classes (resource needs) and not in terms of an entire queue. So the main job container related to priority was not the queue but the job class. This attribute prevented properly running ESP-2, since all the jobs are the same class and priority for the Z test jobs could not be set.

In running ESP-2 with LSF, several configuration changes had to be made to properly complete the test. These changes affected the order in which the scheduler selected jobs to be run. By default, LSF runs in FIFO (First In First Out) order — first job in, first job out. The configuration change indicates to LSF to schedule based on job size — the larger the CPU count needed, the sooner the job needed to be run. The other configuration change needed was the amount of time LSF

waited for enough resources to be collected before it gave up on trying to schedule a job. We increased that value to be greater than the runtime of the longest job.

During the course of the evaluation, Platform sent several different schedulers to test. One was a plug-in scheduler for 5.0 version of the software that ran jobs based on size. This was different from the previous configuration, in that it overcame the resource wait limit. Another version of the software, v5.1, was released that supported third-party schedulers, including the Maui scheduler. LSF worked with the third-party scheduler in testing, but several features of LSF were lost when using it in this way.

OpenPBS was installed on Alvarez by the hardware vendor. OpenPBS is a dynamically configured system set up with a program called qmgr (queue manager). Any changes in the configuration program take effect immediately. The scheduler used with OpenPBS in this evaluation was the Maui Scheduler, as supplied by the system vendor because it provided better support than the stock schedulers in OpenPBS for the type of scheduling needed at NERSC. ESP-2 identified scalability issues because OpenPBS polls all nodes for state,

so a single crashed node could stop scheduling across the entire system. This has since been rectified.

Using ESP-2 for Evaluation

Initial testing indicated none of the software systems passed. Sometimes they failed because ESP-2 required various attributes that the scheduling software did not have at the time. Other failed runs were because the system simply could not withstand the load ESP-2 put upon them. Failures were generally due to the deficiencies pointed out above, which kept ESP-2 from running as intended, or running at all. MauiME, with its lack of submission priorities and lack of a method to allow node exclusivity could not run ESP-2 at all. SGE always ran the full configuration jobs (Z tests) last, regardless of its priority and any amount of configuration.

OpenPBS had the advantage that it came pre-configured on Alvarez and already used the Maui Scheduler. There was no tuning involved (beyond what had been done when it was first installed), so OpenPBS was used to set a baseline.

The default installation of LSF did not complete ESP-2. The full configuration job (Z test), like SGE, would run it last regardless of its

priority. However, when informed of this, Platform Computing added a scheduling module that allowed proper insertion of the Z job. This enabled LSF to properly complete ESP-2. After further tuning, LSF was able to obtain an efficiency rating comparable to OpenPBS with the Maui Scheduler. During the course of the investigation, Platform released a new version of LSF that supported the Maui Scheduler which worked well with LSF. Together, LSF and Maui produced ESP-2 results almost identical to those made by LSF with a modified stock scheduler. Table G-2 compares the ESP-2 results in terms of an effectiveness rating as defined above.

		PBS	LSF	LSF w/mod	LSF w/Maui	SGE	Maui /ME
64 CPUs	Throughput	83.7	75.2	78.6	79.5	n/a	n/a
	Z-Job	77.5	65.0	73.5	73.7	n/a	n/a
128 CPUs	Throughput	83.5	74.8	n/a	n/a	n/a	n/a
	Z-job	78.0	64.3	n/a	n/a	n/a	n/a

Table G-2: ESP-2 Effectiveness Rating of different scheduling systems operating on the same hardware configuration. Note that N/A indicates the test was not successfully completed.

Several of the Z test ESP-2 runs in Table E-2 are depicted graphically in Figures G-1 through G-4. All graphs have been normalized to use the same time scale and to use the same colors and legends when possible. The graphs show node/CPU usage, the

number of running jobs, and the number of queued jobs. The queued jobs run out before the test ended; so no batch scheduler got 100% efficiency in this test. A flat line develops along the top of the “CPU in Use” line; again, the total number of CPU’s was limited to either 64 or 128 depending on the test. Z test shows up in the graphs as the deep canyons in the CPU line, as the schedulers husband resources in order to the start the full configuration jobs by letting the queue run dry.

LSF initially had the problem of wanting to schedule all the small jobs first, so when the scheduler attempted run the Z test, it had to wait for many small jobs to complete before continuing. Switching LSF to large-job first scheduling resulted in an improvement in effectiveness, but not quite equaling OpenPBS/Maui’s effectiveness. Discussions with Platform’s engineers indicate this difference was the result of 2 possible variables – scheduler interval (how often the scheduler is ran) and job interval (how often jobs are started up). There was not time to explore the relationship of these variables while the hardware was available.

Figure G-4 shows how scheduling changes effectiveness. The CPU count data from 3 different ESP-2 runs is graphed. One run is LSF with default scheduling run, another is LSF with the scheduler modification

discussed above, and the third is a LSF with Maui. LSF with Maui achieved the highest efficiency; however, LSF with the scheduler modification was not far behind it

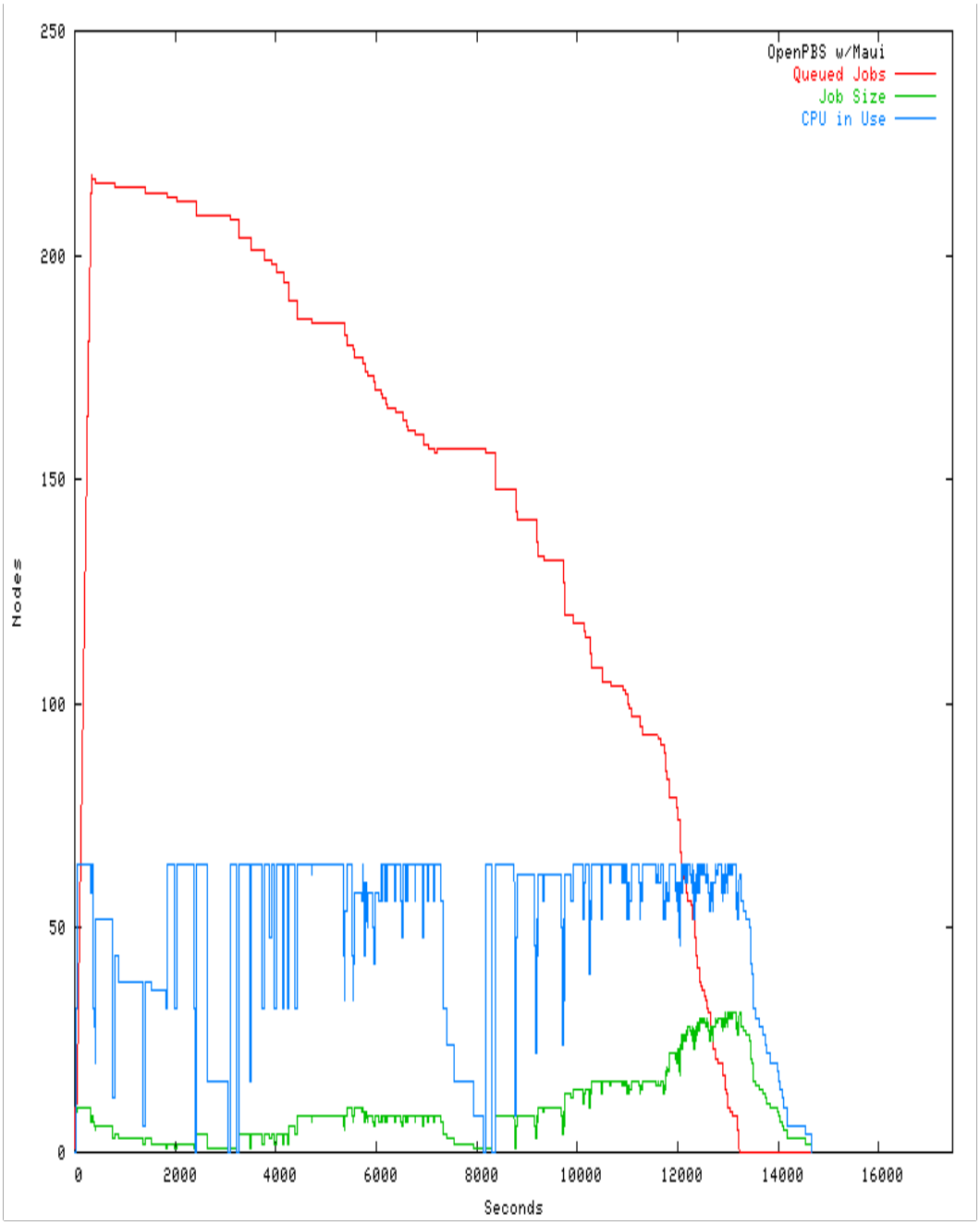


Figure G-1: OpenPBS with Maui shows an ESP-2 rating of 77.5%.

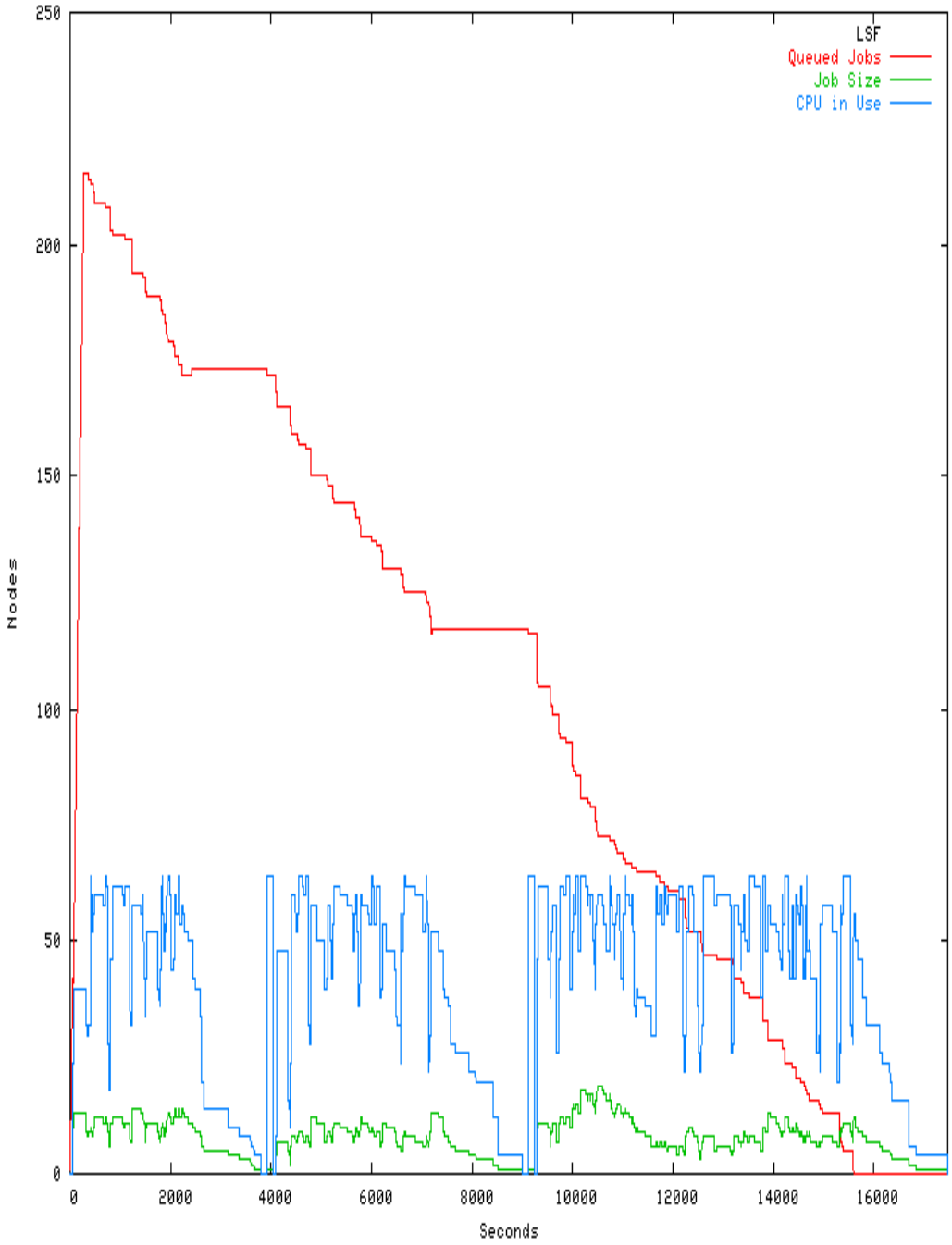


Figure G-2: Basic LSF shows an ESP-2 rating of 65%

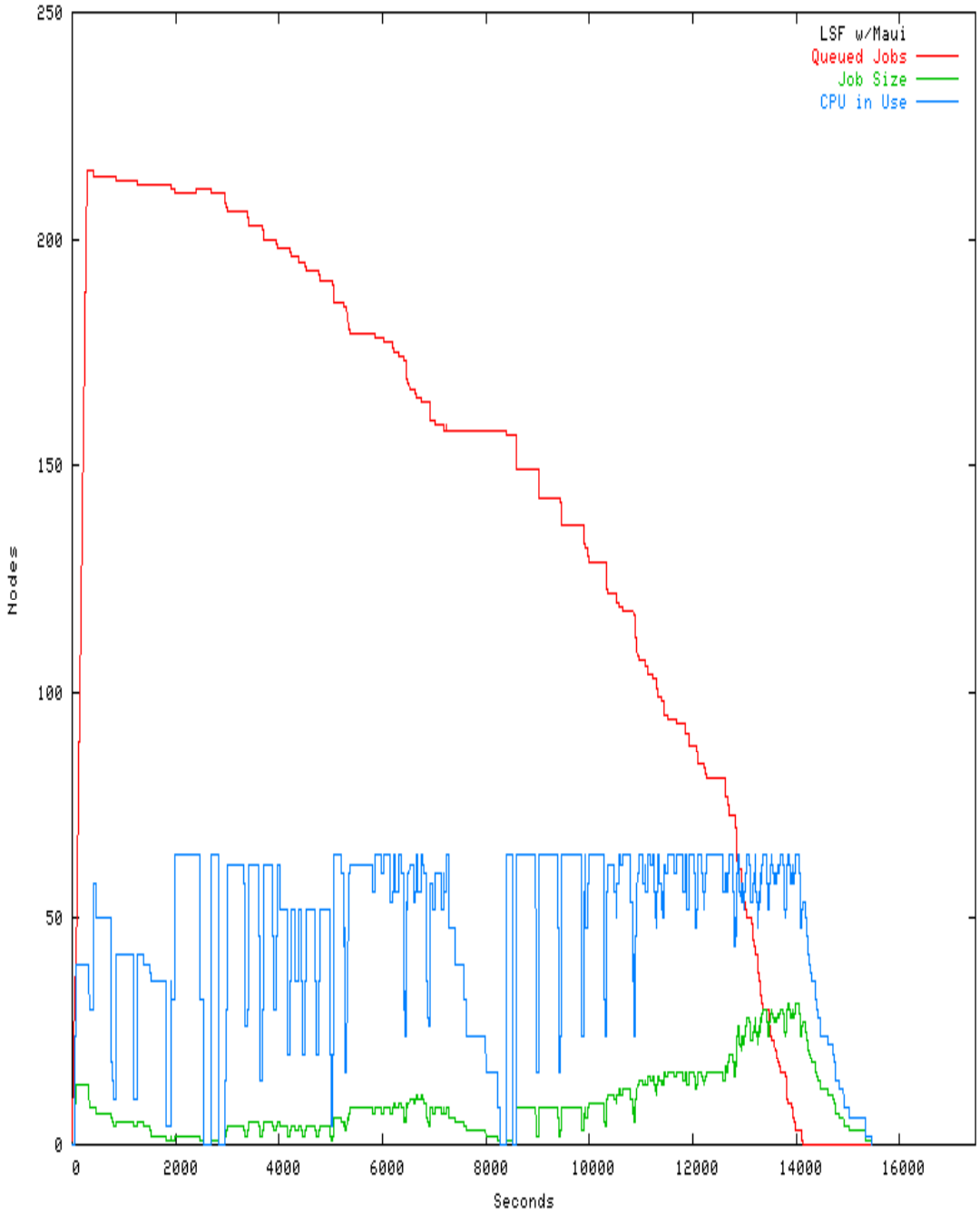
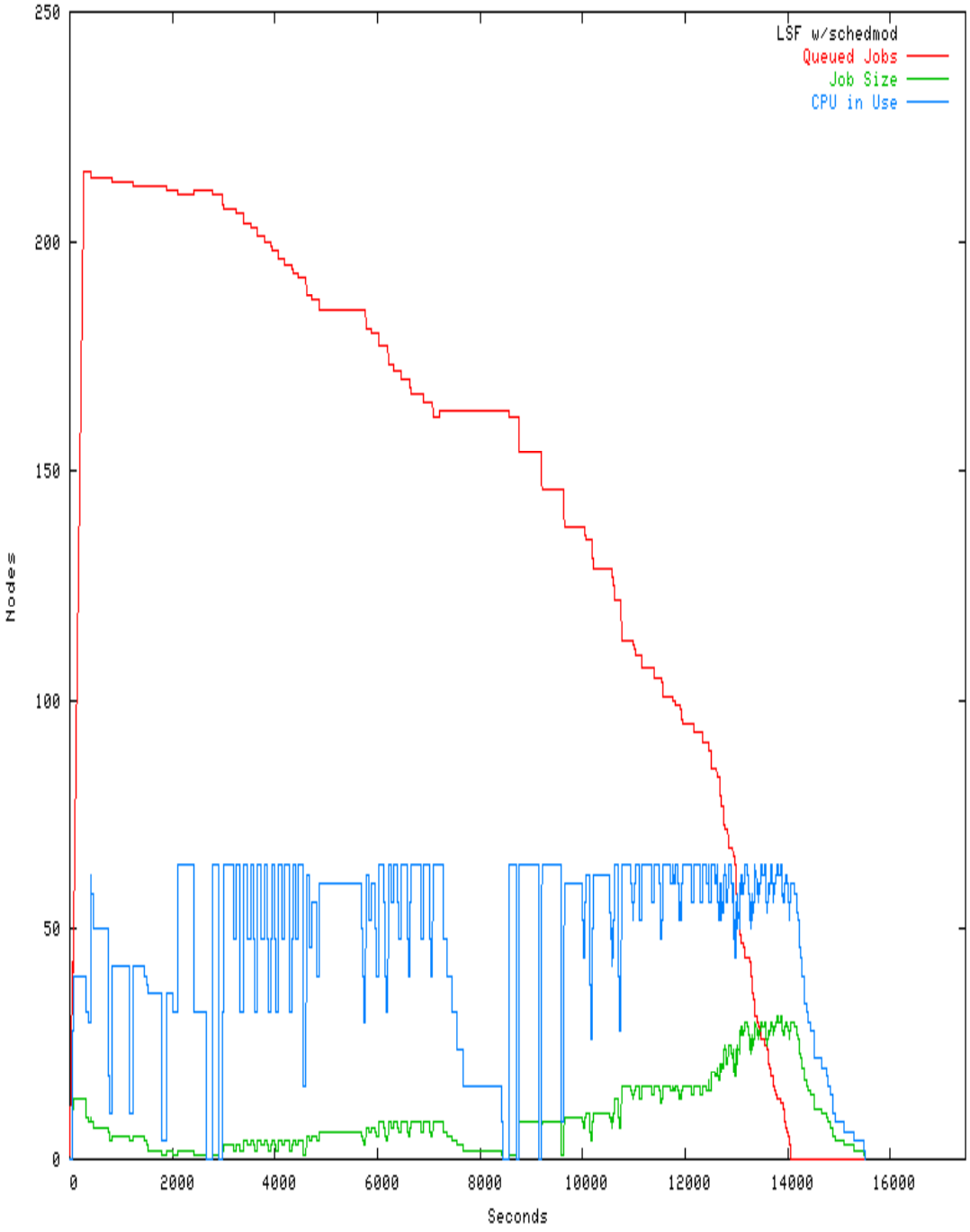


Figure G-3: LSF with the Maui Schedule has an effectiveness of 73.7%.



FigureG-4: LSF with improved scheduling functionality has an ESP-2 rating of 73.4%

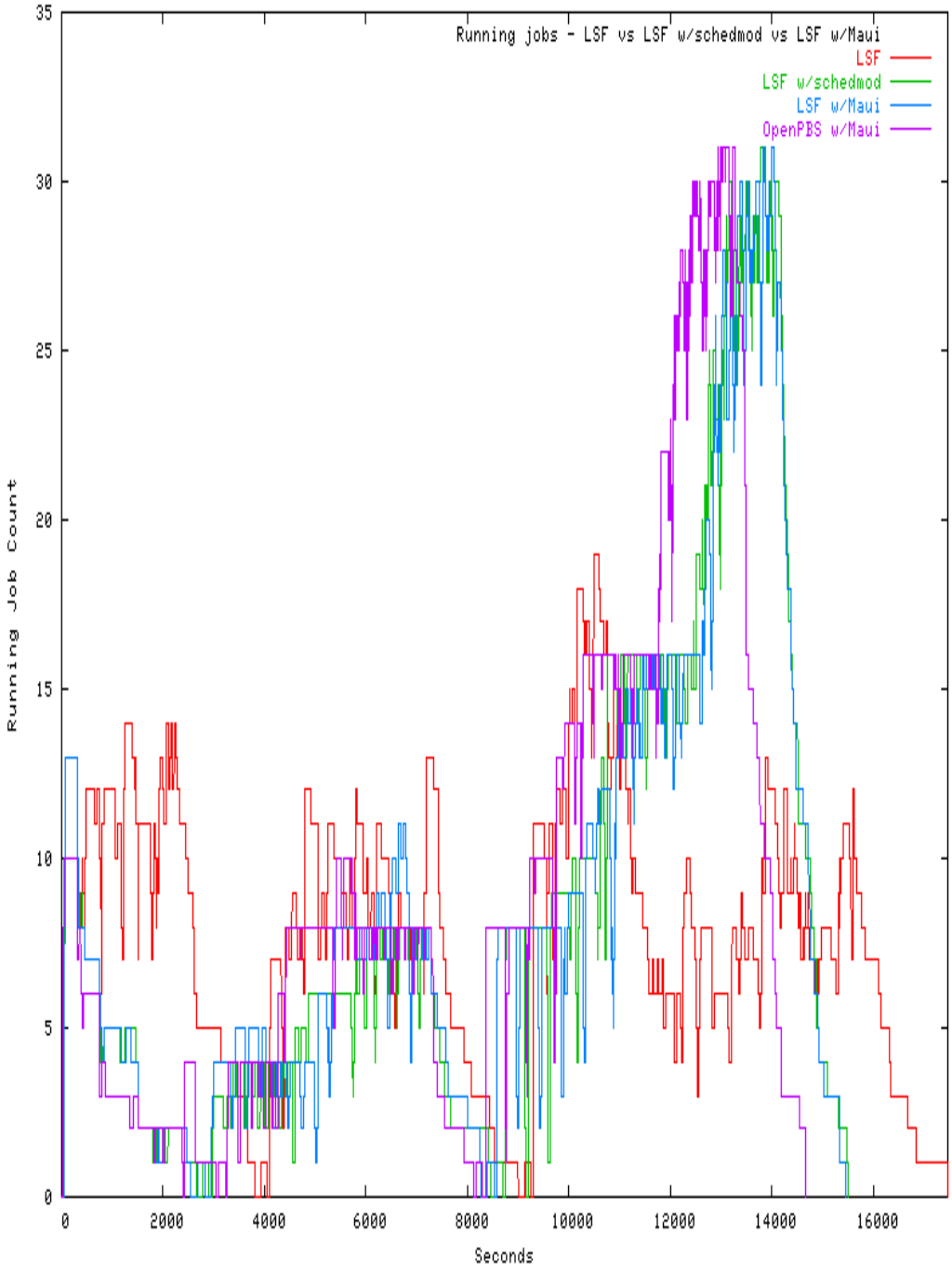


Figure G-5: The count of the number of CPUs busy for ESP-2 runs with four different schedule implementations. The implementation are a) basic LSF (red), b) LSF with modifications to improve running large jobs (green), c) LSF with the Maui Scheduler (blue), and d) OpenPBS with the Maui scheduler (purple)

Assessment Conclusions

The scheduler assessment using ESP-2 took longer than initially planned because many of the systems lacked of robustness and features for parallel- scheduling on Linux platforms. The effort involved to tune the schedulers and explore combinations of features was significant in order to successfully complete ESP-2. This involved exploring documentation, code, and/or on-going communication with vendors. Some software, such as MauiME, was still evolving software. Others, such as FBSNG or SLURM, did not fulfill the needs specific criteria for the test or the NERSC environment. Time was spent working around these software attribute limitations. SGE and MauiME, in particular, responded to the feedback by providing the feature.

Effort was spent comparing the various modes of LSF. LSF was capable of duplicating OpenPBS with Maui's effectiveness score. Most schedulers today are polling based, not interrupt based (i.e., they do not wait for a message of job completion, but keep checking). The interval for the poll impacts the effectiveness. The interval chosen also determines how much CPU time is used – both on the scheduling CPUs and the computational CPUs. The shorter the interval is better

for effectiveness, at the cost of higher CPU loads and potentially decreased consistency.

Although the scheduler assessment project did not result in a clear recommendation, it demonstrated that ESP-2 is a well-defined method for evaluating such software.

Appendix H. Comparing Light Weight Operating Systems (LWOS)

This section briefly discusses other interesting results from the comparison of CLE and CVN on the XT-4 and other tests that are used at NERSC for evaluating systems not directly related to the SSP discussion in Chapter 4: above.

These observations are based on data collected in the summer and fall of 2007, with the last data point being in mid October 2007. At the time, Catamount (the single core reference) and CVN had been in service for close to four years and had been through many cycles of improvements and tuning. It had also run probably thousands of applications at scale even though most were on single core nodes in XT-3 systems. At the time of this study, CLE was not yet released for general use, having exited development at the start of the study and being barely four months in use on only NERSC-5 by the time the last results in this study were observed. NERSC-5 was the first large scale exposure of CLE, and there are many areas that are known to be able to benefit from tuning and further improvement. The fact such robust testing and quality of the results were feasible for a very early operating system was encouraging. That being said, now here are other observations about CLE and CVN.

The average runtimes of the seven medium size application problems were slightly slower on CLE than on CVN, whereas several of the large and extra-large applications were faster. This combined with observations of early user science applications during the evaluation period indicated codes may scale somewhat better on CLE. This conjecture was previously discussed in the Wallace paper on the design goals of CLE referenced above, but this was the first data showing the conjecture may be true. Whether this is due to improved message handling in the node rather than the master-slave CVN is not clear. However, latency using the multi-pong test was not that much different between the two kernels. Intra-node latency was about 30% higher for CLE, but the maximum inter-node latency for the entire system was essentially identical.

CLE had significantly lower streams of memory performance than CVN due to the Linux memory manager. This was particularly true when only 30% of the memory was occupied. However, the streams memory rate had less impact on applications than might be expected because of compiler cache handling. Initially, several NPBs were impacted negatively on CLE, but could be tuned with straightforward

methods to match CVN performance. Full applications, needed little or no tuning to address memory performance difference.

I/O performance differed between CLE and CVN for the IOR benchmark and also for metadata. Lustre version 1.4.6 was used for the file system software for both CLE and CVN. For IOR (IOR Download n.d.), CVN did better for aggregate I/O in the initial assessment, while CLE did much better for single stream performance. The aggregate performance of CLE has since improved. Meta data performance on CLE was somewhat better than CVN.

The average Coefficients of Variation (CoV) across the SSP-4 applications was .4% for CLE and .35% for CVN – remarkably close considering CLE was derived from a full operating system. The CoV was calculated for each SSP-4 application by doing multiple runs of the same application and problem set, and then these individual CoV's were averaged. The low variability of CLE was not predicted as there was concern that increased OS jitter using Linux would decrease consistency.

Finally, the ESP-2 test, described in Chapter 5, ran on CLE, but never completely executed on CVN within the evaluation time period. The traditional through-put test (submitting a set of applications to over

subscribe the batch job scheduler) on NERSC-5 showed less the 1% difference between the two LWOSs.

Bibliography

Allinea. *DDT.* http://www.allinea.com/index.php?page=48.

Altair Grid Technologies, LLC. *PBS Pro User Guide 5.4.* http://www.hipecc.wichita.edu/PBS/PBSProUG_5_4_0.pdf: Altair Grid Technologies, LLC, 2004.

Amber. http://ambermd.org/ and http://amber.scripps.edu/amber9.bench2.html.

American Heritage Dictionary – on-line . http://education.yahoo.com/reference/dictionary/entry/fair.

Amos, Bob, Sanjay Deshpande, Mike Mayfield, and Frank O'Connell. *RS/6000 SP 375 MHz Power3 SMP High Node.* IBM White Paper, IBM, 2001.

Amos, Bob, Sanjay Deshpande, Mike Mayfield, and Frank O'Connell. *S/6000 SP 375 MHz Power3 SMP High Node.* IBM White Paper, IBM Corp.

Bailey, David H., and el.al. "The NAS Parallel Benchmarks." *Intl. Journal of Supercomputer Applications* 5, no. 3 (Fall 1991): 66-73.

Bailey, David. "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers." *Supercomputing Review*, August 1991: 54-55.

Bailey, David, et al. *The NAS Parallel Benchmarks.* Technical Report, NAS Research Branch, NAS (NASA Advanced Supercomputing Division), Moffett Field, CA: NASA Ames, March 1994.

Barrios, Marcel, et al. "Scientific Applications in RS/6000 SP Environments." IBM Redbook, December 1999.

Beecroft, Jon, David Addison, Fabrizio Petrini, and , Moray McLaren. "QsNetII: An Interconnect for Supercomputing Applications and Quadrics High Performance Interconnect." *HotChips 2003.* 2003.

Bender, Carl, et al. Hardware interface between a switch adapter and a communications subsystem in a data processing system. Ed. IBmMCorporation. Patent 6111894. August 26, 1997.

Blakeborough, Jay, and Mike Welcome. "T3E Scheduling Update." *41st Cray User Group Conference Proceedings.* 1999.

Boehm, Barry. "A view of 20th and 21st Century Software Engineering." *ICES 2006.* Shangai, China, May 20-24, 2006.

Borrill, Julian. "MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package." *Proceedings of the 5th European SGI/Cray MPP Workshop.* Bologna, Italy, 1999.

Boucher, Kevin, Brian Femiano, Sara Prochnow, and Allen Peppler. "The Cray X1 Supercomputer." March 2004. http://209.85.173.104/search?q=cache:KS0GbZfN9IMJ:https://users.cs.jmu.edu/abzugcx/public/Student-Produced-Term-Projects/Computer-Organization-2004-SPRING/Cray-X1-by-Sara-Prochnow-Kevin-Boucher-Brian-Femiano-Allen-Peppler-2004-Spring.doc+Cray+X1E+Data+She (accessed 2008).

Bucher, Ingrid, and Joanne Martin. *Methodology for Characterizing a Scientific Workload.* Technical Report, Los Alamos, NM 87545: Los Alamos National Laboratory, 1982.

Camp, William. "State of HPC." Portland, Oregon: Intel HPC Forum, June 2008.

Canning, A., L. W. Wang, A. Williamson, and A. Zunger. "Parallel empirical pseudo-potential electronic structure calculations for million atom systems." *J. Computational Physics*, no. 160:29 (2000).

Carrington, Laura, M. Laurenzano, Allan Snavely, Roy Campbell, and Larry Davis. "How Well Can Simple Metrics Represent the Performance of HPC Applications?" *SC 05 - The High Performance Computing, Storage, Networking and Analysis Conference 2005.* Seattle, WA: Association of Computing Machinary (ACM), 2005.

CFDR. 2006. http://cfdr.usenix.org/.

CNet. "CNet." *IBM Wins hybrid supercomputer deal.* September 6, 2006. http://news.com.com/IBM+wins+hybrid+supercomputer+deal/2100-1006_3-6112975.html (accessed 2006).

Collins, W. D., et al. "The formulation and atmospheric simulation of the Community Atmosphere Model: CAM3." *Journal of Climate* 19 (2006): 2144-2161.

Condor. http://www.cs.wisc.edu/condor/.

Cray Research, Inc. *Cray Assembler for MPP Reference Manual.* Cray Cesearch, Inc., 1996.

—. *Cray T3E System Support Skills.* Mendota Heights, MN: Cray Research, Inc., May 1997.

—. *UNICOS/mk Resource Administration.* 2.0.2. Mendota Heights, MN: Cray Research, Inc., 1996.

"Cray X1E." *Cray X1E Datasheet.* http://www.cray.com/downloads/X1E_datasheet.pdf (accessed 2007).

Cray, Inc. *Cray XT(TM) Series Programming Environment User's Guide.* Section 4.2 CNL Programming Considerations, Cray, Inc., 2007.

Cray, Inc. "Cray XTTM Series Programming Environment User's Guide." Section 4.3 Catamount Programming Considerations, Cray, Inc., 2007.

Culler, David E., and Jaswinder Pal Singh. *Parallel Computer Architecture: A Hardware/Software Approach.* First. Edited by Denise P.M. Penrose. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1999.

Dagum, Leonardo, and Ramesh Menon. "OpenMP: An Industry-Standard API for Shared-Memory Programming." *IEEE Computational Science & Engineering,* (IEEE Computer Society Press) 5, no. 1 (January 1988): 46-55.

Danis, C., and C. A. Halverson. "he value derived from the Observational Component in an Integrated Methodology for the study of HPC Programmer Productivity." *Workshop on Productivity and Performance in High-End Computing. 12th International Symposium on High-Performance Computer Architecture.* Austin, TX, February, 2006.

DeRose, Luiz, and John Levesque. "Tools and Techniques for Application Performance Tuning on the Cray XT4 System." *Tutorial at the 2007 Cray User Group Conference.* Seattle, WA, May 7-10, 2007.

Dicks, Stanley. "Mis-Usability: On the Uses and Misuses of Usability Testing." *SIGDOC ' 02.* Toronto, Ontario, Canada, October 20-21, 2002.

"Dictionary.com." http://dictionary.reference.com/search?q=potency (accessed 2008).

Digitial Review. "At the speed of light through a PRISM." *Digital Review*, December 1998: 39-42.

Dongarra, Jack. "Performance of Various Computers Using Standard Linear Algebra Software in a Fortran Environment." *HPCC.* http://www.netlib.org/benchmarks/performance.ps (accessed 2007).

Dongarra, Jack. *Performance of Various Computers Using Standard Linear Equations Software.* Computer Science , University of Tennessee, Knoxville TN, 37996: University of Tennessee, 1985.

Earl, Joseph, Christopher G. Willard, and Debra Goldfarb. "IDC Workstations and High-Performance Systems Bulletin." *HPC User Forum: First Annual Meeting Notes.* 2000.

FBSD. http://www.freebsd.org/doc/en/books/handbook/.

FBSNG. http://www-isd.fnal.gov/fbsng/.

Figueira, Silvia M., and Francine Berman. "Modeling the Effects of Contention on the Performance of Heterogeneous Applications." *Proceedings of the High Performance Distributed Computing (HPDC '96).* 1966. 392.

Flemming, Philip J., and John J. Wallace. "How not to lie with statistics: the correct way to summarize benchmark results." *Communications of the ACM* (Association for Computing Machinary) 29, no. 3 (March 1986).

Fox, Armando, Emre Kician, and David Patterson. "Combining Statistical Monitoring and Predictable Recovery for Self Management." *Workshop on Self-healing systems, Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems.* Newport Beach, CA, 2004. 49-53.

Gonzalez, Antonio, Scott Mahlke, Shubu Makherjee, Resit Sendag, Derek Chiou, and Joshua Yi. "Reliability: Fallacy or Reality." *IEEE Micro*, November-December 2007: 36-45.

Gould, John, and Clayton Lewis. "Designing for Usability: Key Principles and What Designers Think." *Communications of the ACM* (Association for Computing Machinery) 28, no. 3 (March 1985): 300-311.

Govindaraju, Rama, et al. "Architecture and Early Performance of the New IBM HPS Fabric and Adapter." *Proceedings of the Conference on High Performance Computing (HiPC 2004).* Heidelberg: Springer Berlin, 2005.

Graham, Susan, Mark Snir, and Cynthia Patterson, . *Getting Up to Speed: The Future of Supercomputing, Report of the National Research Council fo the National Academies of Science, , http://research.microsoft.com/Lampson/72-CSTB-.* National Reseach Council of the National Academies of Science, 2004.

Gschwind, M., P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazak. "A novel SIMD architecture for the Cell heterogeneous chip-multiprocessor." *Hot Chips 17.* Palo Alto, CA, 2005.

Helin, J., and K. Kaski. "Performance Analysis of High-Speed Computers." *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing.* Reno, NV: Association for Computing Machinery, 1989. 797 – 808.

"High Performance Technology Insertion 2006 (TI-06) ." *DOD Modernization Program.* 2005. http://www.fbodaily.com/archive/2005/05-May/08-May-2005/FBO-00802613.htm (accessed 2005).

Hockney, Roger W. "The Science of Computer Benchmarking." *SIAM.* Society for Industrial and Applied Mathematics, 1996.

Hofstee, P. "Power Efficient Architecture and the Cell Processor." *High Performance Computing Architectures - 11.* 2005.

Hoise. http://www.hoise.com/primeur/00/articles/monthly/UH-PR-01-00-1.html.

Holbrock, Karen, and David E. Shaw, . *Accelerating Innovation for Competitive Advantage: The Need for Better HPC Application Software Solutions.* Council for Competitiveness, 2005.

Hornbaek, Kasper. "Meta-Analysis of Correlations Among Usability Measures." *CHI 2007.* Portland, OR, April 28-May 3, 2007.

HPCS. http://www.highproductivity.org/.

HPCS. http://www.darpa.mil/ipto/programs/hpcs/index.htm.

IBM, Inc. "HATS and HAGS in RSCT: PSSP V3.5 for AIX Diagnosis Guide." IBM Publication.

IBM, Inc. "IBM System Cluster 1350 Facts & Features Report." June 2008.

IOR Download. https://computing.llnl.gov/?set=code&page=sio_downloads.

ISO. *Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability.* International Standards Organization, 1998.

ISO Usability. http://www.humanfactors.com/downloads/usabilityISO.pdf.

John, Lizy Kurian. "More on finding a Single Number to Indicate Overall Performance of a Benchmark Suite,." *ACM SIGARCH Computer Architecture News* (Association for Computing Machinery) 31, no. 1 (March 2004).

John, Lizy Kurian, and Lieven Kurian, . *Performance Evaluation and Benchmarking.* 6000 Broken Sound Parkway, NW, Suite 300, Boca Raton,, FL, 33487-2742: CRC Press Taylor and Francis Group, 2006.

Jones, James Patton, ed. *Altair PBS Pro™ User Guide for UNIX, Linux and Windows.* Altair Grid Technologies, 2004.

Keen, Noel. "Assessment of Applying the PMac Prediction Framework to NERSC-5 SSP Benchmarks." LBNL Technical Report, NERSC, Lawrence Berekeley National Laboratory, Berkeley, CA, 2006.

Kramer, William, and Clint Ryan. *Performance Variability of Highly Parallel Architectures.* LBNL Technical Report, Berkeley, CA: Lawrence Berkeley National Laboratory, May 2003.

—. "Performance Variability on Highly Parallel Architectures." *International Conference on Computational Science 2003.* Melbourne Australia and St. Petersburg Russia, 2003.

Kramer, William, et al. "Report of the Workshop on Petascale Systems Integration for Large Scale Facilities." LBNL Technical Report, Lawrence Berkeley National Laboratory, 2007.

Lascu, Octavian, Zbigniew Borgosz, Josh-Daniel S. Davis, Pablo Pereira, and Andrei Socoliuc. "An Introduction to the New IBM Eserver pSeries High Performance Switch." IBM Redbook, 2003.

Laudon, James, and Daniel Lenoski. "System Overview of the SGI Origin 200/2000 Product Line." *IEEE Computer Society International Conference: Technologies for the Information Superhighway.* Santa Clara, CA: IEEE-CS Press, February 25-28, 1996.

Lavoie, Richard. *It's So Much Work to Be Your Friend.* New York, NY, Touchstone (Simon and Schuster), 2005.

Lee, C. B., Y. Schwartzman, J. Hardy, and A. Snavely. "Are user runtime estimates inherently inaccurate?" *10th Workshop on Job Scheduling Strategies for Parallel Processing.* New York, NY, 2004.

Li, Xiaoye S. Li, and James W. Demmel. "SSuperlu–dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems." *ACM Trans. Mathematical Software* (Association for Computing Machinery) 29, no. 2 (June 2003): 110-114.

Lilja, David. *Measuring Computer Performance: A Practitioner's Guide.* Cambridge University Press, 2000.

Lin, Z., S. Ethier, T. S. Hahm, and W. M. Tang. "Size scaling of turbulent transport in magnetically confined plasmas." *Physical Review Letters* 88 (2002).

LINPACK Download. 2008. http://www.netlib.org/LINPACK (accessed 2008).

Mashey, John R. "War of the Benchmark Means: Time for a Truce." *ACM SIGARCH Computer Architecture News* (Association for Computing Machinery) 32, no. 4 (September 2004).

Maui. http://mauischeduler/sourceforge.net/.

McMahon, F. *The Livermore Fortran Kernels: A computer test of numerical performance range.* Technical Report , Lawrence Livermore National Laboratory, Livermore, CA: University of California,, 1986.

Membench. http://www.sdsc.edu/pmac/projects/index.html.

Moore, Gordon E. "Cramming more components onto integrated circuits." *Electronics Magazine*, 1965.

MPI Forum. "Document for a Standard Message-Passing Interface." Technical Report, University of Tennessee, Knoxville, TN, 1993.

MVAPICH Ping-Pong Benchmark. https://mvapich.cse.ohio-state.edu/svn/mpi-benchmarks/branches/OMB-3.1/osu_latency.c (accessed 2008).

Nagaraja, Kiran, Xiaoyan Li, Ricardo Bianchini, Richard Martin, and Thu D. Nguyen. "Using fault Injection and Modeling to Evaluate the Performability of Cluster Based Services." *Proceedings fo teh 4th USENIX Symposium on Internet Technologies and Systems.* Seattle, WA, March 2003.

"NERSC SSP Project." *NERSC Projects.* 2004. http://www.nersc.gov/projects/ssp.php (accessed 2004).

"NERSC User Survey." 2003. http://www.nersc.gov/news/survey/2003/.

NERSC User Survey. 2006. http://www.nersc.gov/news/survey/.

"NERSC-5." *NERSC.* 2004. http://www.nersc.gov/projects/procurements/NERSC5 (accessed 2005).

Nielson, Jakob, and Jonathan Levy. *Communications of the ACM* (Association for Computing Machinery) 37, no. 4 (April 1994): 66-75.

Oliker, Lenoid, Julian Borrill, Jonathan Carter, David Skinner, and Rupak Biswas. "Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms." *International Conference on Parallel Processing: ICPP.* 2005.

Oliker, Lenoid, Rupak Bisaw, Rob van der Wijngaart, and David Bailey. "Performance Evaluation and Modeling of Ultra-Scale Systems." Edited by Michael A. Heroux, Padma Raghaven and Horst D. Simon. *Parallel Processing for Scientific Computing* (SIAM), 2006.

Ostle, Bernard. *Statistics in Research.* Ames, IA: The Iowa State University Press, 1972.

PAL. http://www.c3.lanl.gov/pal/index.shtml (accessed 2007).

Pancake, Cheri. *Improving the Quality of Numerical Software Through User-Centered Design.* Technical Report, Lawrence Livermore National Laboratory, June 1998.

PARAllel Total Energy Code. http://www.nersc.gov/projects/paratec.

Patterson, David A., and John L. Hennessy. *Computer Organization and Design – The Hardware/Software Interface.* Third. Burlington, MA: Morgan Kaufmann Publishers, 2007.

Patterson, David, and John Hennessey. *Computer Architecture – A Quantitative Approach.* Second. Burlington, MA: Morgan Kaufmann Publishers, 1996.

PBS. http://www.pbsgridworks.com and http://www.pbsgridworks.com/DocumentationInfo.aspx.

PDSI. 2007. http://www.pdl.cmu.edu/PDSI/FailureData/index.html.

PDSI Failure Data. 2007. http://www.pdl.cmu.edu/PDSI/FailureData/index.html (accessed 2008).

Pinheir, E., W. D. Weber, and L. A. Barroso. "Failure Trend in a Large Disk Drive Population." *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07).* San Jose, CA, 2007.

Pinheiro, Eduardo, Wolf-Dietrich Weber, and Luiz Andre Barroso. "failure Trends in a Large Disk Drive Population." *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07).* San Jose, CA, February 2007.

PMaC. http://www.sdsc.edu/PMaC/.

"Private communication with Mr. Brent Gorda, LLNL." 2005.

"Private communication with Mr. Cray Henry, Director of the Department of Defense (DOD) High Performance Computing Modernization Program (HPCMP)." March 2008.

"Private Communication with Mr. Steve Luzmoor, Cray Inc." 2008.

PSIW. 2007. http://www.nersc.gov/projects/HPC-Integration/.

"PSNAP." 2006. http://www.c3.lanl.gov/pal/software/psnap/README.

RAD Lab. http://radlab.cs.berkeley.edu/wiki/RAD_Lab.

ROCS. http://roc.cs.berkeley.edu/.

RTMAP. 2008. https://rmtap.llnl.gov/abstract.php.

Rubin, H. *Handbook of Usability Testing.* New York, NY: Wiley, 1994.

Sauro, Jeff, and Erika Kindlund. "A Method to Standardize Usability Metrics Into a Single Score." *CHI 2005.* San Jose, CA, April 2-7, 2005.

Schmidt, M. W., et al. "General Atomic and Molecular Electronic Structure System." *Journal of Computational Chemistry*, no. 14 (1993): 1347-1363.

Schroeder, Bianca, and Garth A. Gibson. "Disk Failure in the Real world: What does an MTTF of 1,000,000 hours mean to you?" *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST '07).* San Jose, CA, 2007.

Schroeder, Bianca, and Garth A. Gibson. "Understanding Disk Failure Rates: What does an MTTF of 1,000,000 hours mean to you?" *ACM Transactions on Storage (TOS)* 3, no. 3 (October 2007).

—. "Understanding failures in Petascale Computers." *Journal of Physics Confernece Series.* 2007.

Scott, Steven L., and Gregory M. Thorson. "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus." *HOT Interconnects IV.* Stanford University, CA, 1996.

Selby, Samuel M., ed. *CRC Standard Mathematical Tables Sixteenth Edition.* 18901 Canwood Parkway, Cleveland, Ohio 44128: The Chemical Rubber Co., 1968.

SGE. http://gridengine.sunsource.net/.

Simard, Angèle. "Canadian Meteorological Center Computing Update." *2003 Computational Atmospheric Sciences Workshop.* 2003.

Simon, Horst, and Erich Strohmaier. *Statistical Analysis of NAS Parallel Benchmarks and LINPACK Results.* Vol. 919, in *Lecture Notes In Computer Science*, edited by Bob Hertzberger and Guiseppe Serazzi, 626 - 633. London: Springer-Verlag, 1995.

Sinharoy, B., R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. "POWER5 system micro architecture." *IBM Journal of Research and Development* (IBM, Inc.), Jul-Sep 2005.

Skinner, David, and William Kramer. "Understanding the Causes of Performance Variability in HPC Workloads." *2005 IEEE International Symposium on Workload Characterization (IISWC-2005).* Austin, TX, October 6-8, 2005.

SLRUM. https://computing.llnl.gov/linux/slurm/.

Smith, J. E. "Characterizing Computer Performance with a Single Number." *Communications of the ACM* (Association of Computing Machinary) 31, no. 10 (October 1988): 1202-1206.

Snir, Mark, and David Bader. "A Framework for Measuring Supercomputer Productivity." *International Journal of High Performance Computing Applications* (Sage Publications, Inc.) 18, no. 4 (November 2004): 417.

SPEC Benchmarks. 2000. http://www.spec.org (accessed 2008).

Srinivasan, Srividya,, Rajkumar Kettimuthu, Vijay Subrarnani, and P. Sadayappan. "Characterization of Backfilling Strategies for Parallel Job Scheduling." *nternational Conference on Parallel Processing Workshops (ICPPW'02).* 2002. 514.

"SSP Project Page." *NERSC.* 2008. http://www.nersc.gov/projects/ssp.php (accessed 2008).

Streams Benchmark. http://www.cs.virginia.edu/stream/ (accessed 2008).

Taylor, Mark, Richard Loft, and Joseph Tribbia. "Performance Of A Spectral Element Atmospheric Model (Seam) On The HP Exemplar

2000." *Journal of Scientific Computing* 15, no. 4 (December 2000): 499-18.

The MIMD Lattice Computation (MILC) Collaboration. http://www.physics.indiana.edu/~sg/milc.html and http://www.physics.utah.edu/~detar/milc/.

Tikir, M., L. Carrington, E. Strohmaier, and A. Snavely. "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations." *Proceedings of SC07.* Reno, NV: Association of Computing Machinery (ACM), 2007.

Top 500 List. 2008. http://www.top500.org (accessed 2008).

Ujfalussy, B., et al. "High performance first principles method for complex magnetic properties." *Proceedings of the ACM/IEEE SC98 Conference.* Orlando, FL: IEEE Computer Society, Los Alamitos, CA 90720-1264, November, 1998.

Williams, Dr. Margret. *Private Communication (email).* 2007.

Wong, Adrian, Leonid Oliker, William Kramer, Teresa Kaltz, and David Bailey. "Evaluating System Effectiveness in High Performance Computing Systems." LBNL Technical Report, 1999.

Worley, Pat, and John Levesque. "The Performance Evolution of the Parallel Ocean Program on the Cray X1." *Proceedings of the 46th Cray User Group Conference.* 2004.

Wu, C., and T. Fend. "On a class of multistage interconnection networks." *IEEE Transactions on Computing* 29, no. C (August 1980): 694-702.

Xu, Wei, Joseph L. Hellerstein, Bill Kramer, and David Patterson. "Control Considerations for Scaling Event Processing." *The 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM'05).* Barcelona, Spain, 2005.

Xu, Wei, Peter Bodik, and David Patterson. "A Flexible Architecture for Statistical Learning and Data Mining from System Log Streams." *Workshop on Temporal Data Mining: Algorithms, Theory and Applications at The Fourth IEEE International Conference on Data Mining (ICDM'04).* Brighton, UK, November 2004.

Zhang, Y., A. Sivasubramaniam, J. Moreira, and H. Franke. "mpact of Workload and System Parameters on Next Generation Cluster Scheduling Mechanisms." *IEEE Transactions on Parallel and Distributed Systems* 12, no. 9 (September 2001): 967-985.