What's Working in HPC: Investigating HPC User Behavior and Productivity

Nicole Wolter
San Diego Supercomputing Center
Michael O. McCracken
San Diego Supercomputing Center
Allan Snavely
San Diego Supercomputing Center
Lorin Hochstein
University of Nebraska, Lincoln
Taiga Nakamura
University of Maryland, College Park
Victor Basili
University of Maryland, College Park

Introduction

High performance computing, as a field, involves a great deal of interdisciplinary cooperation. Researchers in computer science work to push the boundaries of computational power, while computational scientists use those advances to achieve increasingly detailed and accurate simulations and analysis. Staff at shared resource centers enable broad access to cutting edge systems while maintaining high system utilization.

Attempts to evaluate the productivity of an HPC system require understanding of what productivity means to all its users. While each of the above groups use HPC resources, their differing needs and experiences affect their definition of productivity. This, in turn, affects decisions about research directions and policies. Because so much is at stake, measuring and comparing productivity is not to be taken lightly. There have been many attempts to define productivity quantitatively, for example, see Kuck[1] for a definition of user productivity and Kepner[2] for a definition of the productivity of a system.

Our approach avoids the problems involved in trying to quantify productivity and instead defines the productivity of a system in terms of how well that system fulfills its intended purpose. Certainly the intended purpose of an HPC system is not just to stay busy all the time, but instead to deliver scientific results. Working with the San Diego Supercomputer Center (SDSC) and its user community, we have analyzed data from a variety of sources, including SDSC support tickets, system logs, HPC developer interviews, and productivity surveys distributed to HPC users. In order to better understand exactly how HPC systems are being used, and where the best opportunities for productivity improvements are, we have compiled a list of conjectures about HPC system usage and productivity (each originally suggested by experienced researchers in HPC) and have compared these to the usage patterns and attitudes of actual users through four studies. The seven conjectures are as follows:

- *HPC users all have similar concerns and difficulties with productivity.*
- *Users with the largest allocations and the most expertise tend to be the most productive.*
- *Computational performance is usually the limiting factor for productivity on HPC systems.*
- *Lack of publicity and education is the main roadblock to adoption of performance and parallel debugging tools.*
- *HPC programmers would require dramatic performance improvements to consider making major structural changes to their code.*
- *A computer science background is crucial to success in performance optimization.*
- *Visualization is not on the critical path to productivity in HPC in most cases.*

In the discussion that follows, we evaluate each of the conjectures. After summarizing the data sources we used and how we collected them, we present our findings and try to clarify how well each of these beliefs actually stands up to the evidence.

[1] Kuck, D. "Productivity in High Performance Computing," *International Journal of High Performance Computing Applications*, 18: 2004. pp. 489-504.

[2] Kepner, J. "High Performance Computing Productivity Model Synthesis," *International Journal of High Performance Computing Applications*, 18: 2004. pp. 505-516.

Procedure

In this study we address the above conjectures by evaluating consult tickets and job logs and further verifying the preliminary assessments, based on the previously mentioned means, with user surveys and by personal interviews of developers using current TeraGrid Sites. More in depth explanation of the studies conducted can be found at SDSC's Performance Modeling and Characterization (PMaC) web site.[3]

[3] PMaC web site for productivity study materials. PMaC Publications. http://www.sdsc.edu/PMaC/HPCS/hpcs_ productivity.html. Aug. 2006.

The initial assessment, to quantify and qualify HPC productivity, was derived from evaluating the user-submitted help tickets. The ticket sampling included all help tickets submitted to the SDSC help desk from March 2004 to March 2006. These tickets span numerous architectures. The consulting tickets enabled the identification of possible HPC resources productivity bottlenecks.

Because only 307 of the 920 registered users submitted support tickets during the time span we investigated, it was clear that ticket analysis alone did not account for all users. Attempting to include a broader set of users, we looked at system job logs of the SDSC DataStar supercomputer,[4] a 2,368-processor IBM Power4 system. We evaluated high-level trends for all 59,030 jobs run on the DataStar P655 nodes at SDSC from January 2003 to April 2006. The jobs ranged in size from 1 to 128 eight-processor nodes.

[4] SDSC User Services. SDSC DataStar user guide: http://www.sdsc.edu/user_services/datastar/

To further address some of these questions raised in the previous two studies we embarked upon a survey and interviewing campaign. We developed an interview strategy based on available references[5] and examples[6] to avoid common pitfalls in design. The questions included a general background and experience section, a development section, and a development practices and process section. The full interview script is available on the SDSC PMaC web site.[3] This strategy was extended to create a user survey. The survey had two goals; first, to get answers from questions similar to the interview script from a larger sample size, and second, to find potential future interview subjects for further investigation. The full survey is also available on the PMaC web site.[3]

[5] Robson, C. "Tactics -the Methods of Data Collection" in *Real World Research*, Blackwell Publishing, 2002.

[6] Cook, C., Pancake, C. "What users need in parallel tool support: Survey results and analysis," *IEEE Computer Society Press*, pages 40– 47, May 1994.

Analysis

The studies above identified some system and usage trends. This section draws conclusions about the conjectures made in the introduction section of this paper, and discusses how the studies influenced our understanding of, and the implications for, HPC centers and users.

Conjecture 1: *HPC users all have similar concerns and difficulties with productivity.*

While it is clear that users have a wide range of system demands, as seen in the variety of allocation and job sizes, it is not uncommon to assume that as long as users share a system, they share the same challenges staying productive. However, as details of individual usage patterns emerged, we found that not all users experienced the same problems with the same systems and tools, and some problems affected some users more than others.

Three distinct classes of users emerged based on HPC resources utilization, project scale, and the problems encountered. While these classes are clearly related to allocation size, they are defined by characteristics of their system usage.

Marquee Users: Marquee users run at very large scale, often using the full system and stressing the site policies and system resources. For these reasons they will be the most affected by node failures and are generally unable to use the interactive nodes to debug or benefit from backfill opportunities to the queue. We have named this group "Marquee" users because such projects are often used to publicize HPC centers.

Marquee users often have a consultant, or are the consultant, working on the application to improve the performance, to port to a new system or scale to larger numbers of processors. The marquee users are generally represented in our study through the job logs and personal interviews.

Normal Users: The largest class, "Normal" users, tend to run jobs using between 128 and 512 processors. Their problem size is not necessarily limited by the available resources of the systems they use. This greater flexibility in their resource usage, allows them the ability to run on smaller systems that may not be as heavily loaded. Normal users are less likely to have been forced to tune for performance optimization, or to have used performance tools. Normal users are generally represented in our study in the form of user surveys and job logs.

Small Users: Small users have a minor impact on system resources, do not tend to use large allocations, and generally run jobs with fewer than 16 processors, which are often started quickly by backfilling schedulers. In general, small users are learning parallel programming, and their productivity challenges are more likely due to unfamiliarity with the concepts of HPC computing. "Small" users are generally represented in our study in the form of help tickets and job logs.

Clear examples of the differences between the normal and marquee users can be seen in the different responses from the verbal interviews and the survey results. In some cases the interviewees and survey respondents were using the same systems and the same software, but due to different processor counts and memory requirements, their opinions of the productivity impact of system traits were very different.

The interviews with SDSC performance consultants showed that memory or I/O was the primary bottleneck for their codes. However, 8 out of 12 of our survey respondents believed that processor floating-point performance was their top bottleneck. Since some survey respondents were using the same code as our interview subjects, we attribute this discrepancy to the scale at which the marquee users run. In many cases performance bottlenecks will only become apparent when the system resources are stressed. The survey respondents, representing our normal users, report standard production runs requesting 256-512 processors, while interview subjects often use more than 1,024 processors.

The interactive running capabilities of a system provide another opportunity to distinguish user classes. Most survey respondents were able to successfully debug their codes on 1-8 processors running for less than one hour, which is feasible on current interactive nodes. Marquee users, on the other hand, must submit to the batch queue and are subject to long wait times to run very short jobs to reproduce bugs and test fixes. The marquee users expressed their frustration with the lack of on-demand computing for a large number of processors, specifically for debugging purposes.

This implies that different system policies are appropriate for different users. For example, in various phases of a project, resource demands may vary. When major development and

tuning has ceased and production has begun, a policy that would allow the entire system to be reserved could be a major improvement to productivity. One marquee user gave the example that it would take a week running around the clock to get one simulation completed and that repeatedly waiting in the queue is wasteful and can extend this process by at least factor of four. In contrast, another researcher was uncomfortable with having dedicated time due to the risk of wasting allocation while fixing problems, preferring a high-priority queue reservation policy instead. System design and site policies should reflect the different types of users and stages of development.

Conjecture 2: ***Users with the largest allocations and most experience are the most productive.***

A common policy of HPC centers is to give preference in queue priority to jobs with higher node requirements, which encourages users to make full use of rare high capacity systems. Large users may also receive more personal attention, even to the point of having a dedicated consultant working on their programs. Does this conclusively imply that larger users are the most productive?

Through our interviews it emerged that productivity, in terms of generating scientific results, is just as difficult to achieve for large users, if not more so, than for smaller users. Queue wait time, reliability and porting issues all cause major problems for large users. Large-scale programs often push the limits of systems and therefore run into problems not often seen at lower scale, such as system and code performance degradation and system reliability problems.

Evaluating the queue based on job logs can be complicated. There are a number of factors that could affect queue priority. Some of the most influential variables are length of runtime, processor count requested, the size of a user's unspent allocation, as well as site administrators' ability to change the priority of individual jobs directly.

Processor Count	Average	Median	Maximum	Minimum
8	8.87	0.06	858.93	0.00
64	12.64	0.45	792.33	0.00
128	24.12	2.56	752.67	0.00
256	21.13	3.51	415.69	0.00
512	19.79	3.81	266.44	0.00
1024	23.99	9.07	205.49	0.00

Table 1. Wait time of jobs 2003-06 on DataStar, in hours.

Processor Count	Average	Median	Maximum	Minimum
8	2.43	0.52	624.89	0.00
64	2.49	0.69	49.42	0.00
128	3.84	0.63	105.01	0.00
256	3.33	0.46	100.06	0.00
512	3.13	0.40	19.44	0.02
1024	2.60	0.31	18.01	0.03

Table 2. Run time of jobs 2003-06 on DataStar, in hours.

Processor Count	Average	Median	Maximum	Minimum
8	8.02	1.15	22666.83	1.00
64	19.22	1.66	11826	1.00
128	68.71	3.31	5137.17	1.00
256	44.42	5.35	3809.40	1.00
512	53.53	6.00	3924.50	1.00
1024	90.69	11.68	2800.39	1.00

Table 3. Expansion factor for jobs 2003-06 on DataStar.

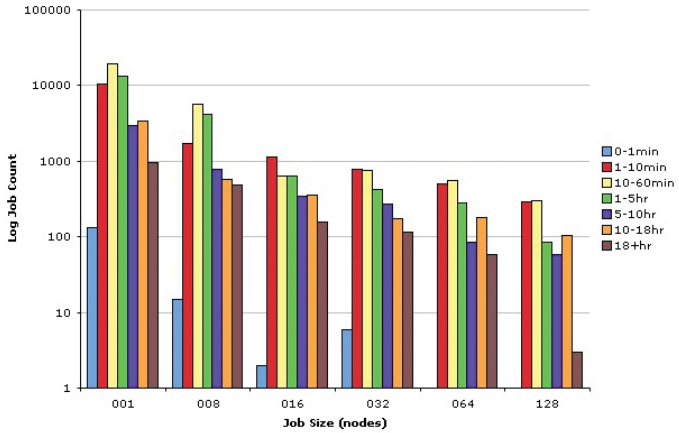

Figure 1. Run timedistribution, grouped by job size.

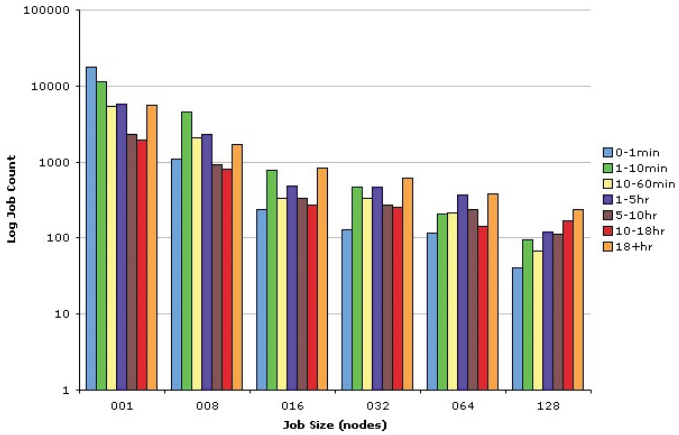

Figure 2. Wait time distribution, grouped by job size.

In support of our initial statement, Table 1 displays some trends that are not surprising. On average small jobs have shorter wait times. The average wait time for jobs requesting only one node was nine hours while the average wait time for 128 nodes (1024 processors) was approximately 24 hours. In contrast, the inverse trend is evident in maximum wait times; as the number of nodes increased, the maximum wait time decreased. The maximum wait time for a 128 node job was 17 days and the maximum wait time for one node was 71 days. The median tells us a similar story to the average. The trends in run time and wait time are also visible in the

histograms shown in Figure 1 and Figure 2, grouped by job size. Table 2 shows statistics for the job run time.

The expansion factor ((waittime + runtime) / runtime) data in Table 3 reinforces the trends established by the wait times and also displays how the queue favors large allocations. The average expansion factor increases as the number of nodes increases up to 256 processors, where there is a dip in the expansion factor curve. This appears to show a preference for large jobs in the scheduler beginning at 256 processors. On closer inspection, the trend was influenced by a single high priority account with a large allocation, running many jobs on 256 processors. The actual priority increase for "large" jobs begins at 512 processors or more. Also note the high expansion factor for 1024 processor runs. While it would seem to show that users running on large portions of the system are not very productive, we see that this effect is due to a large number of such jobs with very short run time. 632 of the 839 jobs that ran on 128 nodes (1024 processors) ran for less than two hours, and most of those jobs were part of a benchmarking effort. The average expansion factor for jobs that were not benchmarking was 9.58, and the median was 10.05, which more clearly shows the scheduler favoring large jobs as intended.

Although the analysis above does show that marquee users receive priority for using large portions of the system and are therefore not disproportionately hurt by queue wait time, they do face a serious productivity bottle-neck in the inability to get a large-enough allocation on one system, or on the right system for their task. Due to site policies, single-site allocations often have a limit. In some cases, users were forced to port codes to several systems due to allocation limits, resource constraints, or system and support software compatibility issues. One developer told us of having to port one program to four different systems during the period of one project. Another told us of having to port to at least two systems for an individual run to be able to capitalize on the different features supplied on different systems.

Porting itself can be time consuming and difficult. We heard about the time required to complete a port, ranging from less than an hour to many days, and in some cases the successful port was never achieved. Once the code is ported, these multi-platform scenarios tend to require tedious manual effort from the user to move data files around, convert formats, and contend with the problems of working on many sites with different policies, support, and capabilities.

Conjecture 3: *Time to solution is the limiting factor for productivity on HPC systems.*

While the initial migration of a project to HPC systems is certainly due to expanding resource requirements, we found across all four studies that the HPC users represented in our samples treat performance as a constraint rather than a goal to be maximized. Code performance is very important to them only until it is "good enough" to sustain productivity with the allocation they have, and then it is no longer a priority. In economics literature, this is called satisficing,[7] and while it is not surprising in retrospect, it is important to keep this distinction in mind when thinking about what motivates HPC users. The in-depth system log evaluation showed most users not taking advantage of the parallel capacity available to them. Out of 2,681 jobs run in May 2004, 2,261 executed on one eight-processor node. This accounted for 41% of all the CPU hours utilized on the system. Furthermore, the job logs show that in between 2004 and 2006, 1,706 jobs out of 59,030 jobs on DataStar were removed from the batch nodes for exceeding the maximum job time limit of 18 hours. Of these jobs, 50% were running on fewer than eight processors. Given access to a resource with thousands of processors, the majority of users choose to reduce the priority of performance tuning as soon as possible, indicating that they have likely found a point at which they feel they can be productive.

[7] Simon, H. "Rational choice and the structure of the environment," *Psychological Review*, Volume 63: 2002. pp. 129–138.

The support ticket evaluation tells a similar story. Many users requested longer run times, and fewer than ten users requested anything related to performance. Furthermore, some of the users requesting longer run times did not have checkpoint restart capabilities in their code, and many were running serial jobs.

While performance is certainly the original reason for moving to an HPC platform, the attitudes and statements of our interview subjects reinforces the assertion that it is not a primary goal of theirs. When asked about productivity problems, performance was generally taken for granted while other issues such as reliability, file system capacity and storage policies, as well as queue policies and congestion, were discussed in depth. Performance problems are just one of many barriers to productivity, and focus on performance at the expense of other improvements should be avoided.

Conjecture 4: ***Lack of publicity is the main roadblock to adoption of performance and parallel debugging tools.***

While it is true that many users are unfamiliar with the breadth of performance tools available, a significant portion of users simply prefer not to use them. Reasons given by interviewees included problems scaling tools to large number of processors, unfamiliar and inefficient GUI interfaces, steep learning curves, or the overwhelming detail provided in the displayed results.

Unsurprisingly, the performance optimization consultants were the most likely to use tools. However even they often opted for the seemingly more tedious path of inserting print statements with timing calls and recompiling rather than learning to use performance tools.

The situation for parallel debuggers was no more encouraging. Only one interviewee used readily available parallel debugging tools. However, other developers indicated that if the debugging tools were consistent and easy to use at scales of up to hundreds of processors, they would have been used. In their current state, parallel debugging tools are considered difficult or impossible to use by the interviewees who had tried them.

It is clear that there is a long way to go before common HPC programming practice embraces the powerful tools that the research community has built. Certainly there is a lack of motivation on the part of many HPC programmers to learn new tools that will help them with the task of performance optimization, which is not always their main priority. Aside from the obvious issue of acceptable performance at large scale, it seems that continuing to strive for tools that are easier to learn and use is important for improved adoption. As discussed by Pancake,[8] it is important to design tools from a user's perspective and with early user involvement in the design process for the design to be effective.

[8] Pancake, C. "Can Users Play an Effective Role in Parallel Tool Research?" in *Tools and Environments for Parallel Scientific Computing*, SIAM, 1996.

Conjecture 5: ***HPC programmers would demand dramatic performance improvements to consider major structural changes to their code.***

Programmers we surveyed showed a surprising indifference to the risks involved in rewriting often very large code bases in a new language or changing a communication model. Although many successful projects last for tens of years without making significant changes, eight of the 12 Summer Institute attendees responded that they were willing to make major code changes for surprisingly small system performance improvements or policy changes.

Many of the codes discussed were larger than 100,000 lines of code (LOC). Though all eight codes over 10,000 LOC had checkpoint restart capabilities, three users were willing to rewrite code for the ability to run a single job longer, two requesting only a factor of two job runtime limit extension. The respondents were likely unaware that most sites will allow such small exceptions on a temporary basis. Of the three respondents with more than 100,000 lines of code, only one considered such a major change, requesting in return a factor of ten improvement in either processor speed or job time limits. The results imply that although performance, judged by time to solution, is not always the main goal of HPC users, users would be very receptive to work for guaranteed system and policy changes.

While we see some conflicting responses, it might be possible for HPC centers to capitalize on these attitudes to get users to use profiling tools and spend some effort to improve individual code performance, and ultimately queue wait times, by giving minor compensation to cooperative users. Also, in some cases, it would seem that there is a gap between what users want and what they think they can get. This gap could be bridged with improved communication with users regarding site policies and resource options.

Conjecture 6: ***A computer science background is crucial to success in performance optimization.***

It may seem straightforward to say that if you want to improve your code performance you should take it to a computer scientist or software engineer. However, of the developers on successful projects interviewed, only one had a formal computer science background. In fact, many of these successful projects are operating at very large scale without any personnel with a formal computer science or software engineering background.

There was a general consensus among interviewees that without competence in the domain science, changes to the project are destined for failure. Two subjects actually contracted out serious code bugs and made use of library abstractions to avoid in depth knowledge of parallel programming. One such library was written by a computer scientist, and once it matured, users were able to achieve high performance using it without a parallel computing background.

HPC centers and software developers should keep in mind that their target audience is not computer scientists, but rather physical scientists with a primary focus on scientific results.

Conjecture 7: ***Visualization is not on the critical path to productivity in HPC.***

Every interview subject and all but three survey respondents used visualization regularly to validate the results of runs. For those projects, any problems and productivity bottlenecks that affect visualization have as much impact on overall productivity as poor computational performance during a run.

Such bottlenecks can include problems with available visualization and image conversion software and with finding capable resources for post-processing and visualization rendering. As evidence, one interviewee stated a desire for a large head node with a large amount of RAM to handle rendering. This implies that users' productivity could be greatly enhanced by a dedicated visualization node that shared a file system with computation nodes.

Conclusion

In performance tuning, it is common to hear advice to profile before attempting an optimization, because often the true bottleneck is a surprise. In this paper, we have attempted to find where the true bottlenecks in HPC productivity are by investigating general trends in user behavior and by asking users directly.

Our analysis shows that, in general, HPC user needs are heterogeneous with respect to HPC resource usage patterns, requirements, problems experienced and general background knowledge. These factors combined dictate how individual productivity is viewed and clarifies the motivations for focusing on performance. Our research also gave us insight into the tools and techniques currently available and to what extent they have been embraced in the community.

Future research will evaluate the effectiveness of some of the techniques discussed by users. Based on our findings, we will continue to evaluate HPC community feedback to build a general consensus on the problems that affect productivity and where research time and system money can best be spent to allow these systems to live up to their promise and continue to support leading-edge research.

Acknowledgments

The authors thank the subjects of our interviews and surveys for their cooperation. We also thank the staff of SDSC user services, Douglass Post, and Nick Wright for help coordinating various parts of the study and for their insight.

CTWatch QUARTERLY

ISSN 1555-9874 VOLUME 2 NUMBER 4A **NOVEMBER 2006**

High Productivity Computing Systems and the Path Towards Usable Petascale Computing

GUEST EDITOR **JEREMY KEPNER**

SC2006 PRINT EDITION

INTRODUCTION

FEATURED ARTICLES

Available on-line at http://www.ctwatch.org/quarterly/

CyberInfrastructure Partnership

Cyberinfrastructure Technology Watch

http://www.ctwatch.org/