User-Guided Symbiotic Space-Sharing of Real Workloads

Jonathan Weinberg
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093-0505
jonw@sdsc.edu

Allan Snavely
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093-0505
allans@sdsc.edu

ABSTRACT

Symbiotic space-sharing is a technique that can improve system throughput by executing parallel applications in combinations and configurations that alleviate pressure on shared resources. We have shown prototype schedulers that leverage such techniques to improve throughput by 20% over conventional space-sharing schedulers when resource bottlenecks are known. Such evaluations have utilized benchmark workloads and proposed that schedulers be informed of resource bottlenecks by users at job submission time; in this work, we investigate the accuracy with which users can actually identify resource bottlenecks in real applications and the implications of these predictions for symbiotic space-sharing of production workloads. Using a large HPC platform, a representative application workload, and a sampling of expert users, we show that user inputs are of value and that for our chosen workload, user-guided symbiotic scheduling can improve throughput over conventional space-sharing by 15-22%.

Keywords

symbiotic space-sharing, job scheduling, high performance computing, resource allocation

1. INTRODUCTION

The preponderant model of resource splitting on today's Supercomputers is space-sharing, a model by which executing jobs receive exclusive use of some number of processors as requested by their submitters. Exclusive use of processors however, does not imply exclusive use of the machine's other resources. Depending on the architecture, processors may share caches, memory banks, file systems, or other resources among themselves. On *DataStar* [3] for example, the San Diego Supercomputer Center's production IBM Power4 system, the two processors on each chip share an L2 cache, the four chips on each p655 node share an L3 cache, main memory, an on-node file system, and bandwidth to off node I/O, WAN access, and interconnect.

Because processors share resources, one may infer that concurrently executing processes will affect each other's performance. If some processors make heavy use of a shared cache for example, the level of service delivered to each would likely be inferior to that had fewer processors been stressing that cache.

Due to the nebulous understanding of these inter-application effects, scheduling policies may attempt to minimize resource sharing by allocating resources to jobs in minimally overlapping bundles. DataStar for instance, allocates resources by the node, ensuring that jobs only share global off-node resources that are less likely to present significant contention-related slowdown. While such a policy achieves fairness and predictability, it often achieves sub-optimal performance as well. A CPU or I/O-bound job that is not memory-intensive for instance, may extract very little utility from the memory resources allocated to it. Memory-intensive jobs may require minimum node counts just to fit into memory or may choose to increase memory resources by running across more nodes than their processor counts demand; consequently, some processors can be left idle or lightly loaded.

Symbiotic space-sharing is the idea that schedulers can achieve more efficient resource allocations by executing applications in *symbiotic* combinations and configurations that decrease contention for and increase utilization of shared resources. We have previously shown that given a stream of parallel benchmarks and the limiting resource of each, a prototype symbiotic scheduler can increase throughput by 20% over traditional scheduling techniques [30].

While those results are promising, it is essential to verify that symbiotic space-sharing is as effective for real applications as for smaller, specialized benchmarks whose resource bottlenecks are known and often intended. Is there as much opportunity for symbiotic space-sharing in production workloads as in benchmarks?

Further, while previous work has suggested that the limiting resource of each application be identified by the user at submission time, the accuracy with which users can supply this information for real applications is unknown. How well do users know their applications and is this knowledge accurate enough to inform scheduling decisions?

In the following section, we detail the motivating opportunity space for symbiotic space-sharing based on benchmark workload results. In Section 3, we describe a usage model for symbiotic space-sharing and evaluate it using a production application workload and a sampling of relevant users. We close with related and future works in Sections 4 and 5 respectively.

2. SYMBIOTIC SPACE-SHARING

In this section we present the motivating opportunity for symbiotic space-sharing by quantifying the effects of resource contention on DataStar's p655 nodes and by demonstrating how this contention can be mitigated by alternative job mixes and configurations.

ICS06, June 28-30, Cairns, Queensland, Australia

2.1 Hardware Environment

The results presented in this section were derived from application runs on the San Diego Supercomputer Center's DataStar. The nodes used are IBM P655+, each consisting of 8 Power4 processors running at 1.5 GHz.

Each POWER4 processor contains a 32 KB L1 data cache. Two processors together comprise a *chip* and share a 1.5 MB L2 cache. The L3 cache on each chip is combined with that on the others to create a single node-wide, address-interleaved L3 cache of 128 MB.

Each node is also equipped with 16 GB of memory and a local scratch file system of approximately 64GB. Nodes are directly connected to the GPFS (IBM's parallel file system) through a Fibre Channel link and to each other by the Federation interconnect.

DataStar schedules jobs using a batch queueing model implemented by LoadLeveler [9]. Because the scheduler interface does not allow users to directly request that jobs be coscheduled, we achieved this effect when necessary by deploying MPI jobs that execute the desired sub-jobs on specified processors depending on rank.

2.2 Effects of Resource Contention

Symbiotic space-sharing is predicated on two assumptions: (1) as more processes make use of a shared resource, the level of service provided by that resource to each degrades and (2) applications that make use of different resources can be coscheduled to mitigate such degradation. To demonstrate this concept in practice, we perform a series of tests using single processor runs of the following benchmarks in various combinations. Each is intended to stress a different shared resource:

EP - Embarrassingly Parallel is one of the NAS Parallel Benchmarks [6]. It evaluates an integral by means of pseudorandom trials and is a compute-bound code that makes limited use of shared resources. We use this as a control group to discern between performance degradation in the other benchmarks due to resource sharing and that which is attributable to other overheads.

I/O Bench - A synthetic benchmark that measures the rate at which a machine can perform reads and writes to an arbitrary state of disk by performing sequential, backward, and random read and write tests to file. We use it to stress the on-node file system.

STREAM - A simple synthetic benchmark that measures sustainable memory bandwidth for vector compute kernels by performing a long series of regularly-strided accesses through memory [14, 2]. STREAM is highly cacheable and prefetchable and we therefore use it stress the machine's cache structure.

GUPS - Giga-Updates-Per-Second measures the time to perform a fixed number of updates to random locations in main memory [14, 1]. We use this benchmark to investigate the effects of high demand on main memory bandwidth.

Each graph depicted in Figures 1 through 4 illustrates how increased resource contention affects the overall performance of each benchmark. For each test, we occupy all processors of an 8-way node with some combination of benchmarks.

The Y axis of each graph represents the slowdown incurred by the measured benchmark as more instances of it (X axis) execute concurrently on the same node. Each line, labelled "[PRIMARY] w/ [BACKGROUND]" indicates that the processors not used for running the PRIMARY benchmark executed the BACKGROUND benchmark during the test. We calculate slowdown as $(T_N - T_1)/T_1$ where T_i is the runtime of the benchmark while i instances of it run concurrently on the node. We do not present rigorous error bounds on these experiments because the conclusions we intend to draw need only be notional.

The compute-bound code EP exhibits no sensitivity to increased instances of itself or other benchmarks, indicating that it does not make significant use of shared resources. Contrarily, I/O Bench exhibits super-linear slowdowns when more instances run concurrently on each node. Like EP however, none of the other three benchmarks has any measurable impact on I/O Bench's performance.

The results for the two memory-intensive applications are similar save for when they are coscheduled. Some of the complexities inherent in symbiotic space-sharing are revealed by the one-way interference between GUPS and STREAM. Although GUPS has little effect on the performance of STREAM (Figure 3), the converse is untrue (Figure 4). This one-way interference is due to STREAM's heavy cache use and the relatively low rate of memory operations achieved by GUPS. STREAM increases the L2 and L3 miss rates of GUPS by around .2 each while the presence of GUPS does not affect STREAM's cache miss rates.

These experiments demonstrate that resource contention amongst resource-intensive processes on a node can potentially cause significant slowdown and that this slowdown can be mitigated by alternate job mixes. We have shown these results to be generalizable over a wide range of relevant benchmarks [30].

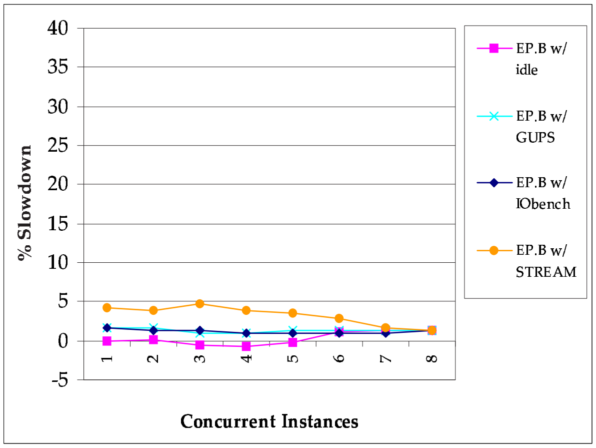

Figure 1: Resource contention effects on EP.

2.3 Scheduling Parallel Applications

In the previous section, we have shown that resource contention among coscheduled processes can cause slowdowns and that such slowdowns can be largely mitigated by symbiotic space-sharing. However, we must still demonstrate that the approach is viable for parallel applications.

Generally, parallel codes employ every processor on each node they use. The scheduler's motivation to use fewer nodes is to minimize the occurrence of slower, inter-node communications in favor of faster, intra-node communications and ostensibly reap performance benefits. The results presented in section 2.2 however, indicate that there is a counter force at play. The processes of parallel applications tend to perform similar operations and consequently

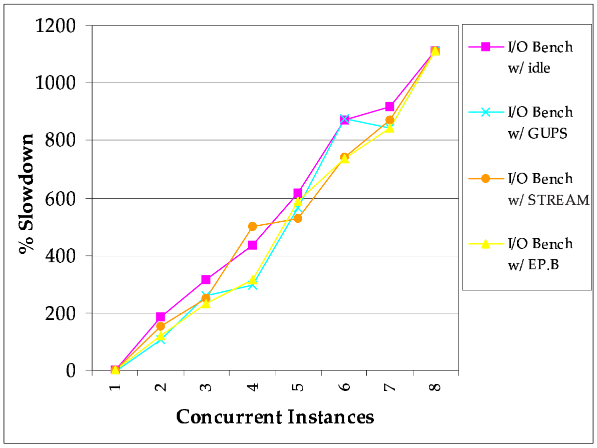

Figure 2: Resource contention effects on I/O Bench.

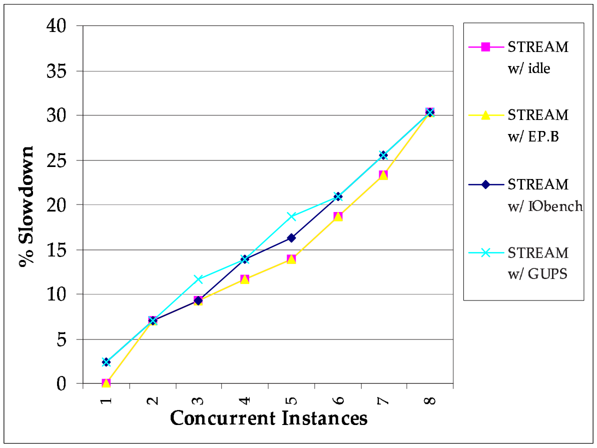

Figure 3: Resource contention effects on STREAM.

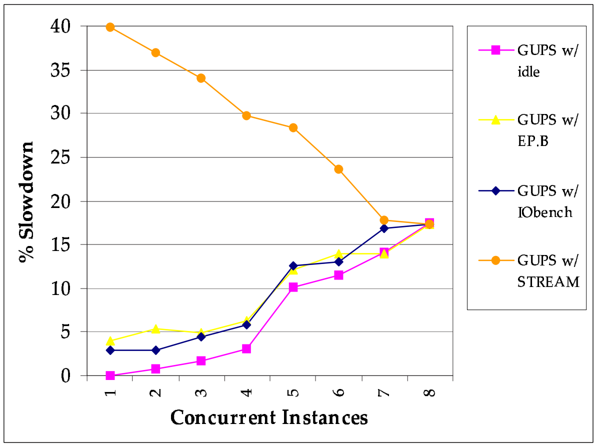

Figure 4: Resource contention effects on GUPS.

Benchmark	Speedup
BT	1.13
MG	1.34
FT	1.27
LU	1.47
CG	1.55
IS	1.12
EP	1.00
BTIO EP	1.16
BTIO SIMPLE	4.97
BTIO FULL	1.16

Table 1: Speedup of 16-processor runs when executing across four nodes instead of two

Bench A	Bench B	Speedup A	Speedup B
CG	IS	1.18	1.17
	BT	1.05	1.04
	EP	1.36	1.03
	BTIO(E)	1.38	1.07
	BTIO(S)	.55	1.03
	BTIO(F)	1.36	1.12
IS	BT	1.04	1.03
	EP	1.07	1.03
	BTIO(E)	1.11	1.07
	BTIO(S)	1.00	2.41
	BTIO(F)	1.13	1.13

Table 2: Speedup attained when parallel benchmarks share four nodes instead of running separately on two each.

stress the same shared resources. Can the performance benefits of symbiotic space-sharing parallel codes outweigh the penalty of additional inter-node communications?

The results in Table 1 suggest that this is indeed the case. We execute two 16-processor runs of each NAS Parallel Benchmark [6], first across two of DataStar's 8-way p655 nodes and then evenly spread across four nodes, effectively utilizing only four processors per node.

We use the MPI implementation of the NPB version 3.2 with problem class C. Because we run these benchmarks across multiple nodes, the I/O tests as performed by variations of BTIO cannot utilize the on-node file system, but rather only the system's network mounted file system GPFS (General Parallel File System). Speedup is calculated using the traditional definition T_2/T_4 where T_N is the runtime of the benchmark on N nodes.

The results reveal that speedup from reduced resource contention in this benchmark set not only outweighs communication overheads, but does so significantly and consistently.

To demonstrate that some speedup is maintained even while the unused processors are not idle, we re-run some of the 4-node tests, but allow two benchmarks to run on the nodes concurrently. For each result presented in Table 2, we execute two parallel benchmarks concurrently on four nodes with each benchmark using exactly half of each node. We consider the BTIO variations to be I/O-intensive, EP to be CPU-intensive, and the rest to be either memory-intensive or resource bottleneck unknown.

These results show that speedup can be maintained even while no processors are idle. Speedup can be induced both by mixing benchmark categories and even by mixing some memory-bound codes. In one instance however, we observe some cross-category slowdown. CG and BTIO_Simple both slow considerably when paired. Nevertheless, these results demonstrate that executing parallel codes in symbiotic combination can indeed yield significant performance benefits. The average speedup increase is 15%, showing that for this set of benchmarks, the benefits of reduced resource sharing outweigh the increased cost of inter-node communications.

2.4 Prototype Scheduler

To test whether or not symbiotic space-sharing can indeed improve system throughput, we implemented a rudimentary symbiotic scheduler to compete against DataStar's production counterpart.

The scheduler was deployed on DataStar and given an ordered stream of one hundred randomly selected 4 and 16-processor jobs to execute using four nodes. We refer to these job sizes simply as *small* and *large*. The jobs in the stream consisted of a four processor version of I/O Bench and variations of the NAS Parallel benchmarks from the following set: {EP.B.4, BT.B.4, MG.B.4, FT.B.4, DT.B.4, SP.B.4, LU.B.4,CG.B.4,IS.B.4,CG.C.16, IS.C.16,EP.C.16, BTIO_FULL.C.16}[1]. The job stream was generated by iteratively enqueueing jobs selected by weighted probability; small jobs were favored over large jobs in a 4:3 ratio and memory-intensive jobs were favored over compute and I/O intensive jobs in a 2:1:1 ratio. Each job was submitted with a synthetically generated expected runtime (within 20% of the actual) that was used by the scheduler for EASY backfilling [13].

The symbiotic scheduler mimics DataStar scheduling objectives by favoring large jobs and backfilling small ones whenever possible. To constrain backfilling opportunities, at most twelve jobs occupy the queue at any given time.

The algorithm employed by the symbiotic scheduler is the most simplistic possible: the scheduler partitions each node evenly into *top* and *bottom* halves. It then executes jobs designated as memory-intensive on the top half and all others on the bottom half. The symbiotic scheduler spreads large jobs across all four nodes while the DataStar scheduler executes each on only two.

DataStar's makespan for the first eighty seven jobs was 5355s while the symbiotic scheduler completed the same jobs in 4451s, a speedup of 1.20. We ignore the final thirteen jobs because the eighty seventh job completed was the final memory-intensive job in the stream and the symbiotic scheduler's memory half was thereafter starved, an artifact of the testing procedure.

3. SYMBIOTIC SPACE-SHARING IN PRACTICE

The prototype scheduler described in Section 2.4 represents the simplest algorithm that can leverage symbiotic properties among applications: coarsely categorize each application by its limiting resource and schedule only cross-category mixes. Even this simplistic approach however, requires that the scheduler be somehow informed of each application's resource bottleneck.

It may be possible to acquire such information by automated experimentation [22, 23] or by correlating profiling statistics with application properties [5, 10, 25, 30]. The most straight forward approach, however, is to simply ask the user.

We envision that initial efforts may implement symbiotic space-sharing as a job submission option. Users who opt to participate might be required to submit the job category (CPU, main memory, or I/O) and the scheduler can act accordingly. In more advanced scenarios, the system may verify the category through profiling techniques or historical run information as has been used for admissions controls [7] or automated runtime estimation [8, 21, 18].

At least three rationales motivate this approach:

Accountability: Any scheduling technique that holds the potential to alter a job's runtime, and consequently its cost, should be explicitly approved by the user. Those uncomfortable with node sharing or those whose jobs require a higher degree of runtime predictability may choose not to participate; those who do participate will more clearly understand and anticipate discrepancies in an application's response times.

Familiarity: Submission flags from users are the standard for communicating job requirements to the scheduler. Including a "limiting_resource" flag alongside the others that users already declare in their submit scripts is practical and familiar.

Simplicity: From a technical perspective, asking the user is simplest and requires the least infrastructure support. In this work we seek to determine the value of such an approach and the degree to which it should be supplemented or replaced by more advanced technical solutions.

In summary, asking users appears to be the shortest path to materializing at least *some* gains from the idea of symbiotic scheduling. However, even this minimally functional implementation is unassured of success. It is unclear how accurately users or administrators can actually determine the resource bottlenecks of real applications. Do such bottlenecks even exist? If they do, then are they well enough pronounced to merit symbiotic space-sharing? Is the idea applicable to production applications at all?

Utilizing a group of expert users at the San Diego Supercomputer Center and a representative application workload, we seek to answer these questions.

3.1 A Realistic Application Workload

The symbiotic space-sharing results we have presented thus far have been derived using benchmark workloads consisting mostly of NAS Parallel Benchmarks with processor counts of up to sixteen. To evaluate the approach's feasibility in production environments, results from larger, more complex codes are necessary.

The following applications were put forward by the National Science Foundation in 2006 for a $30M procurement entitled High Performance Computing System Acquisition: Towards a Petascale Computing Environment for Science and Engineering [4]. All respondents proposing to acquire a supercomputer were required to project their expected performance on these applications. The list is meant to represent supercomputing applications of strategic importance to the United States federal government from the Department of Defense, Department of Energy, and the National Center for Atmospheric Research:

WRF - From the DoD's High Performance Computing Modernization Program, the **W**eather **R**esearch **F**orecasting system has been built with the purpose of serving as the "the nation's next generation mesoscale numerical weather prediction model".

OOCORE - Also from the DoD's HPCMP program, the **O**ut **O**f **C**ore solver is a code developed by the ScaLAPACK project, a collaborative effort in producing high-performance linear algebra routines for distributed-memory MIMD computers. Testing the performance of the out-of-core solver is used

[1] For small jobs, each of the 4 processors actually performs a full serial run of the benchmark at class B with collective communication at the beginning and end.

App.	FLOP	Mem	I/O	Comm. Latency	Comm. Bandwidth
MILC	x			x	
PARATEC	x				x
HOMME		x			x
WRF		x			x
OOCORE			x		

Table 3: Limiting resources of applications as reported by users

to infer the expected performance of the SWITCH code, a partially proprietary electromagnetic solver from Northrop Grumman that spends the bulk of its running time performing OOCORE.

MILC - From the Department of Energy's National Energy Research Scientific Computing (NERSC) program, the **MI**MD **L**attice **C**omputation code is a body of high performance research software written in C for performing four dimensional SU lattice gauge theory simulations.

PARATEC - The **P**arallel **T**otal **E**nergy **C**ode is a NERSC code that performs ab-initio quantum-mechanical total energy calculations using pseudopotentials and a plane wave basis set.

HOMME - The **H**igh **O**rder **M**ethods **M**odelling **E**nvironment, a code from the National Center for Atmospheric Research, is a set of tools to create high-performance scalable global atmospheric models.

Together, this set constitutes a realistic workload from which we can more accurately project the effectiveness of symbiotic space-sharing in production environments.

3.2 What Do Users Know?

We assigned each of the five applications to a different expert user at the San Diego Supercomputer Center. Each user was asked to identify the sensitivity of his application to: (1) FLOP speed, (2) memory bandwidth, (3) I/O bandwidth, (4) communication bandwidth, (5) communication latency. The application experts we interviewed were in many cases cognizant of the instruction mixes and sensitivities because they had at some point written or tuned at least some of their assigned code. To confirm or improve their estimates, users reported to have used MPI Trace, HPMCOUNT, and a proprietary I/O tracer from IBM to collect communications statistics, performance counters, and I/O overheads.

The users' reports are aggregated in Table 3. The resource bottlenecks appear to be rather evenly distributed among our traditional categories and interconnect limitations are common. Are these user guesses accurate enough to inform scheduling decisions?

3.3 User-guided Symbiotic Space-Sharing

To evaluate the usefulness of user inputs in scheduling this workload, we seek to quantify the mutual effects of node-sharing among these applications. To do so, we configure each application to execute on 64 processors with input sizes that produce runtimes between fifteen minutes to two hours on DataStar's 32-way p690 nodes. We execute each application first on its own set of two p690 nodes to establish the baseline performance as represented by the traditional space-sharing approach. We then spread each application across four nodes to determine the effect of an optimal symbiotic space-sharing assignment. Lastly, we measure mutual effects by executing each combination of two applications simultaneously across four nodes.

To gauge the performance of each combination, we instrument the applications with lightweight hardware counters and compare the memory operations per second and floating point operations per cycle achieved by each application. To temper the cost and length of the experiments, only the first fifteen minutes of each run were used.

To assure proper redundancy in our test set, we introduce a second I/O code, a 64-processor run of the NAS Parallel Benchmark BTIO Full at problem class D [32].

Table 4 reports the speedup of each combination as inferred from floating point operations per cycle and approximately confirmed by memory operations per second. The numbers correspond to the speedup of each *primary* application on the Y-axis as it executes across four nodes (utilizing half the processors on each node) while the other half nodes collaborate to execute the *background* application on the X-axis. Speedup is calculated with respect to the primary application's two node run as scheduled by traditional space-sharing.

The numbers in bold red indicate combinations where user inputs have suggested that both primary and background applications are demanding of the same shared resource. Scheduling these combinations risks slowdown to *at least one* of the two applications involved.

The numbers in italic green indicate combinations where user inputs have suggested that the primary application is demanding of a shared resource that is not heavily utilized by the background application. These are predicted to be symbiotic combinations; italic green numbers are therefore expected to be greater than 1.

The gray numbers indicate combinations where user inputs cannot be used to predict any change in performance. The black numbers along the diagonal represent the 2-node runs and are equal to 1 by definition.

The results are noteworthy in several respects. First, as indicated by the *Idle* column, every application exhibits significant speedup when executing on its own across four nodes instead of two, even when communication bottlenecks exist. These results echo those presented in Table 1 where the NAS benchmarks are shown to have the same property. This suggests that for many applications, mitigating resource contention through application spreading may induce benefits that outweigh the costs of increased inter-node communications. This premise underlies symbiotic space-sharing and its apparent generalization to production applications is significant.

Another important result is that user inputs enable us to infer which combinations in this workload will exhibit slowdown. Combining the I/O codes results in mutual slowdown while the memory-intensive combination produces a uni-directional slowdown similar to that demonstrated in Section 2.2 and found commonly in benchmark workloads [30]. The average slowdown of combinations expected to slow is 16.25%.

Conversely, combinations we expect to be symbiotic based on user inputs, yield speedup in all but one case. The only error among sixteen such predictions is the 5% performance drop of OOCORE when run with WRF. Overall, symbiotic combination predictions for this experiment are approximately 94% accurate. The average speedup of such combinations is 15% over traditional scheduling.

These findings imply that a naive user-guided scheduler that runs symbiotic combinations at random, it will achieve a 15% increase in throughput. A more advanced scheduler can experiment with different combinations to eventually discover the best schedule[2]

[2] {(HOMME,PARATEC),(WRF,BTIO),(OOCORE,MILC)}

	Background Benchmark						
	HOMME	WRF	BTIO	OOCORE	MILC	PARATEC	Idle
HOMME	1.00	**1.07**	*1.02*	*1.00*	*1.03*	*1.07*	*1.18*
WRF	**0.68**	1.00	*1.04*	*0.95*	*1.00*	*1.04*	*1.22*
BTIO	*1.13*	*1.18*	1.00	**.90**	*1.16*	*1.20*	*1.32*
OOCORE	*1.05*	*1.25*	**0.70**	1.00	*1.05*	*1.42*	*1.72*
MILC	1.10	1.18	1.11	1.65	1.00	1.21	*1.95*
PARATEC	1.35	1.06	1.20	1.29	1.15	1.00	*2.34*

Table 4: Speedup of Y-axis application while X-axis application is running on other half of the used nodes

and realize a 22.3% improvement. Furthermore, because the worst user-guided schedule[3] still yields an average speedup of 7%, an experimenting scheduler pays no penalty in throughput with respect to traditional scheduling.

4. RELATED WORK

A preliminary evaluation of these techniques has been performed using benchmark workloads [30]. We extend these results by studying their applicability to production workloads and the ability of users to identify resource bottlenecks in real applications.

Many previous investigations of multi-resource aware job scheduling have been conducted, though none under assumptions applicable to today's supercomputing installations. Our approach revisits the issue by starting with a modern production policy on a large MPP machine and relaxing some procedures to achieve higher performance and utilization. We assume rigid job sizes, FCFS-type queued space-sharing, and run-to-completion scheduling with no preemption.

We characterize previous related work into the following non-exclusive categories:

4.1 Multithreading

Symbiotic job scheduling was originally proposed for machines utilizing Simultaneous Multithreading[22, 23], later known as *Hyperthreading*, and was subsequently refined by McGregor et. al. [17]. Such examples are concerned with intimate, cycle-by-cycle resource sharing of multithreaded processors where sharing and contention involve functional units on the processor. Contrastingly, this work focuses on space-sharing contention for off-chip resources by multiple processors.

4.2 Paging

Some studies have sought to schedule job combinations that may limit the amount of paging induced by the workload. In 1994, Peris modelled the cost of paging behavior in parallel applications when working sets would not fit into local memory [20]. Batat and Feitelson suggest limiting the multiprogramming level of gang schedules in order to ensure that job combinations do not exceed a total memory limit [7]. Suh and Rudolf have proposed that if such a limit must be breached, then previously obtained application profile information can inform the scheduler of the best way to do so [25].

Though ensuring a job's ability to fit into memory is encompassed by this work, it is not the sole focus. We address contention for all resources on each node including caches, memory bus bandwidth, and local I/O in addition to global resources shared among multiple nodes. We also study the effects of allocating a job's processes across multiple nodes in order to compare slowdowns from resource contention and inter-node communications.

4.3 Time-sharing

Application-aware job scheduling for time-sharing scenarios has also been studied. Many have proposed *affinity* techniques that mitigate cache perturbations by avoiding process migrations [24, 29, 27]. Such considerations are unique to time-sharing.

Wiseman and Feitelson have suggested that I/O and compute-intensive jobs can be symbiotically coscheduled in a relaxed gang scheduling environment through coordinated preemption points [31]. In contrast to those efforts, this work targets resource sharing and contention in pure space-shared systems.

4.4 SMP Memory Bus Contention

It is well known that contention on the memory bus of an SMP is a scaling bottleneck. Several studies have therefore investigated the possibility of relieving pressure on this bottleneck through intelligent job scheduling. Liedtke introduced the topic in 2000 [12]. Both Antonopoulos [5] and Koukis [10] have built upon this by proposing techniques for scheduling jobs on SMP nodes in a manner cognizant of memory bus contention.

As with the paging example, our study encompasses this concern but is not singularly concerned with it. Further, we extend the discussion from single node benchmark results to full-scale applications across multiple nodes and in a production environment under pure-space sharing. Both previously mentioned works are concerned with gang scheduled servers.

4.5 Other Related Work

Some previous studies of multiple-resource allocation have also been conducted. Parsons and Sevcik investigated the coordinated allocation or processors and memory [19]; subsequently, Leinberger et. al generalized the problem to *k-resource scheduling* where the idea is to choose optimal job working sets when multiple resource requirements exist [11]. Unlike our study, this work assumes independently allocatable resources and well defined requirements for each job. On our target architecture, a predetermined bundle of resources is provided to a job along with each processor.

Also related is work demonstrating that queue times and throughput can be increased if users are asked to submit different cpu-count options for the same job [28]. Our work here does not require different cpu-count options but is somewhat similar in providing more scheduler flexibility with a resulting improvement in throughput and efficiency.

It has been observed that spacing I/O-intensive jobs in time on a parallel file system improves performance [16]. Our emphasis is primarily to spread these in space, and also to identify specific symbiotic partners for such jobs.

Mache and Garg have focused on finding a spatial layout for con-

[3] {(HOMME,OOCORE),(WRF,PARATEC),(BTIO,MILC)}

current jobs in a parallel space-shared machine to minimize communication and maximize access to I/O nodes for I/O-intensive jobs [15]. We address a related but different problem in considering not only the physical layout but also the sets of jobs contending for resources.

Also described has been an approach for deriving beneficial symbiosis (i.e. commensalism), wherein one version of a program, executing concurrently with the main program, helps the main program resolve control-flow for instruction fetching [26]. Alternatively, we search the existing job-stream for sets to co-schedule that interfere as little as possible with each other. We expect the existence of commensal job combinations in realistic production environments to be unlikely.

5. CONCLUSIONS AND FUTURE WORK

In this work, we have shown that symbiotic space-sharing is a promising technique for improving the performance efficiency of large-scale parallel machines in production environments. We have confirmed a key underpinning of the technique by demonstrating that spreading resource-intensive applications across more nodes yields performance benefits that can outweigh increased communication costs. We have extended previous findings on benchmark workloads by demonstrating that symbiotic space-sharing is effective for a representative set of production applications. We have also proposed a usage model for symbiotic space-sharing and shown it to be feasible by demonstrating that users and system administrators are able to inform schedulers of application bottlenecks with sufficient accuracy.

Our results are derived from DataStar, a production machine at the San Diego Supercomputer Center and an application workload put forward by a $30M National Science Foundation procurement as being strategically important to the United States government. For this workload, we have shown that user-guided symbiotic space-sharing can increase throughput by 15-22% over conventional space-sharing.

With the opportunity space mapped, the most important prerequisite for a trial deployment is the development of scheduling heuristics. Such heuristics must be thoroughly evaluated and proposed alongside a framework for understanding and tuning their tradeoffs in the context of disparate environments and application workloads. Interplay between throughput, fairness, and queue times must be transparent enough to inform site-specific policies.

Mechanisms by which schedulers can identify symbiotic mixes in the job stream must also be built. Most simply, such mechanisms can take the form of a job submit option as proposed in this work. A next step could be to allow administrators to configure the scheduler with defaults, and a yet more advanced technique would allow the scheduler to discover symbiotic combinations by acquiring and analyzing profiling data. Each of these solutions must be approached with care as to assure no harm.

We are currently also extending this feasibility study onto grid schedulers in an attempt to understand the degree to which a grid-wide scheduler can improve the efficiency of its resource pool by scheduling symbiotic job combinations at each site. Through this approach, we also hope to study the degree to which a scheduler can increase throughput by lessening site load on resources such as a parallel I/O file system.

6. ACKNOWLEDGEMENTS

We would like to thank Wayne Pfeiffer, Giri Chukkapalli, Robert Harkness, Donald Frederick, and Mustafa Tikir for their participation in this study. This work was supported in part by the DOE Office of Science through the award entitled HPCS Execution Time Evaluation, and by the SciDAC award entitled High-End Computer System Performance: Science and Engineering. This work was also supported in part by NSF NGS Award #0406312 entitled Performance Measurement & Modeling of Deep Hierarchy Systems.

7. REFERENCES

[1] http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/.

[2] http://www.cs.virginia.edu/stream/.

[3] http://www.npaci.edu/DataStar/guide/home.html.

[4] http://www.nsf.gov/pubs/2005/nsf05625/nsf05625.htm.

[5] C. D. Antonopoulos, D. S. Nikolopoulos, and T. S. Papatheodorou. Scheduling Algorithms with Bus Bandwidth Considerations for SMPs. *icpp*, 00:547, 2003.

[6] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[7] A. Batat and D. G. Feitelson. Gang Scheduling with Memory Considerations. In *14th Intl. Parallel Distributed Processing Symp.*, pages 109–114, 2000.

[8] R. Gibbons. A Historical Application Profiler for Use by Parallel Schedulers. In *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 58–77, London, UK, 1997. Springer-Verlag.

[9] S. Kannan, P. Mayes, M. Roberts, D. Brelsford, and J. Skovira. *Workload Management with LoadLeveler*. IBM, November 2001.

[10] E. Koukis and N. Koziris. Memory Bandwidth Aware Scheduling for SMP Cluster Nodes. In *PDP '05: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05)*, pages 187–196, Washington, DC, USA, 2005. IEEE Computer Society.

[11] W. Leinberger, G. Karypis, and V. Kumar. Job scheduling in the presence of multiple resource requirements. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 47, New York, NY, USA, 1999. ACM Press.

[12] J. Liedtke, M. Volp, and K. Elphinstone. Preliminary thoughts on memory-bus scheduling. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 207–210, New York, NY, USA, 2000. ACM Press.

[13] D. A. Lifka. The ANL/IBM SP Scheduling System. In *IPPS 1995 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 949, pages 295–303, 1995.

[14] P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Baily, and D. Takahashi. Introduction to the HPC Challenge Benchmark Suite, April 2005. Paper LBNL-57493.

[15] J. Mache, V. Lo, and S. Garg. Job Scheduling that Minimizes Network Contention due to both Communication and I/O. In *14th International Parallel and Distributed Processing Symposium*, page 457, Washington, DC, USA, 2000. IEEE Computer Society.

[16] J. Mache, V. Lo, M. Livingston, and S. Garg. The impact of spatial layout of jobs on parallel I/O performance. In *IOPADS '99: Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pages 45–56, New York, NY, USA, 1999. ACM Press.

[17] R. L. McGregor, C. Antonopoulos, and D. Nikolopoulos. Scheduling Algorithms for Effective Thread Pairing on Hybrid Multiprocessors. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, CO, April 2005. IEEE Computer Society Press.

[18] A. Mu'alem and D. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. In *12th Intl. Parallel Processing Symposium*, pages 542–546, April 1998.

[19] E. W. Parsons and K. C. Sevcik. Coordinated allocation of memory and processors in multiprocessors. In *SIGMETRICS '96: Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 57–67, New York, NY, USA, 1996. ACM Press.

[20] V. G. J. Peris, M. S. Squillante, and V. K. Naik. Analysis of the impact of memory in distributed parallel processing systems. In *SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 5–18, New York, NY, USA, 1994. ACM Press.

[21] W. Smith, V. Taylor, and I. Foster. Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 202–219. Springer Verlag, 1999.

[22] A. Snavely and D. Tullsen. Symbiotic Job Scheduling for a Simultaneous Multithreading Processor. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 234–244, November 2000.

[23] A. Snavely, D. Tullsen, and G. Voelker. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor. In *Proceedings of the ACM 2002 Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS. 2002)*, pages 66–76, Marina Del Rey, June 2002.

[24] M. Squillante and E. Lazowska. Using Processor-Cache Affinity Information in Shared-Memory Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systemse*, 4(2):131–143, February 1993.

[25] G. E. Suh, L. Rudolph, and S. Devadas. Effects of Memory Performance on Parallel Job Scheduling. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 116–132, London, UK, 2001. Springer-Verlag.

[26] K. Sundaramoorthy, Z. Purser, and E. Rotenberg. Slipstream Processors: Improving both Performance and Fault Tolerance. In *Architectural Support for Programming Languages and Operating Systems*, pages 257–268, 2000.

[27] J. Torrellas, A. Tucker, and A. Gupta. Evaluating the Performance of Cache-Affinity Scheduling in Shared-Memory Multiprocessors. *Journal of Parallel and Distributed Computing*, 24(2):139, February 1995.

[28] G. Utrera, J. Corbal, and J. Labarta. Using moldability to improve the performance of supercomputer jobs Source. *Journal of Parallel and Distributed Computing*, 62(10):1571–1601, October 2002.

[29] R. Vaswani and J. Zahorjan. The Implications of Cache Affinity on Processor Scheduling for Multiprogrammed Shared Memory Multiprocessors. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pages 26–40, Pacific Grove, CA, October 1991.

[30] J. Weinberg and A. Snavely. Symbiotic Space-Sharing on SDSC's Datastar System. In *The 12th Workshop on Job Scheduling Strategies for Parallel Processing*, St. Malo, France, June 2006.

[31] Y. Wiseman and D. Feitelson. Paired Gang Scheduling. In *IEEE Transactions on Parallel and Distributed Systems*, volume 14, pages 581–592, 2003.

[32] P. Wong and R. V. der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical report, NASA Ames Research Center, Moffett Field, CA 94035-1000, January 2003. NAS Technical Report NAS-03-002.