

When Jobs Play Nice: The Case For Symbiotic Space-Sharing

Jonathan Weinberg
jonw@sdsc.edu

Allan Snavely
allans@sdsc.edu

San Diego Supercomputer Center, University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093-0505

Abstract

Using a large HPC platform, we investigate the effectiveness of “symbiotic space-sharing”, a technique that improves system throughput by executing parallel applications in combinations and configurations that alleviate pressure on shared resources. We demonstrate that relevant benchmarks commonly suffer a 10-60% penalty in runtime efficiency due to memory resource bottlenecks and up to several orders of magnitude for I/O. We show that this penalty can be often mitigated, and sometimes virtually eliminated, by symbiotic space-sharing techniques and deploy a prototype scheduler that leverages these findings to improve system throughput by 20%.

1 Introduction

On SDSC’s DataStar [1], as on all parallel systems, processes must share resources. The system does not time-share and consequently, while each process receives its own processor with a level 1 cache, two must share a level two cache. The eight processors on each node must share a level 3 cache, main memory, an on-node file system, and bandwidth to off-node I/O.

Because of this resource sharing, scheduling policies on production space-shared systems attempt to isolate jobs wherever possible. On DataStar, for instance, parallel jobs are never time-shared and each has exclusive use of the nodes on which it runs.

This is not an ideal policy in several circumstances. Resource utilization and throughput suffers when small jobs are forced to occupy an entire node while making use of only a few processors. The policy also encourages users to squeeze large parallel jobs onto the fewest number of nodes possible since doing otherwise is both costly and detrimental to system utilization. Such configurations are not always optimal; the processes of parallel jobs often perform similar computations, stressing the same shared resources and

exacerbating slowdown due to resource contention.

In such situations, a more flexible and intelligent scheduler could increase the system’s throughput by more tightly space-sharing *symbiotic* combinations of jobs that interfere with each other minimally. Such a scheduler would need to recognize relevant job characteristics, understand job interactions, and identify opportunities for non-destructive space-sharing.

The purpose of this study is to investigate the feasibility of such an approach and quantify the extent to which it could improve throughput if implemented. To address these questions, we must determine:

- * To what extent and why do the threads of jobs interfere with each other and those of other applications?
- * If this interference exists, how effectively can it be reduced by alternative job mixes?
- * Are these alternative job mixes feasible for parallel codes and what is the net gain?
- * How can a job scheduler create symbiotic schedules?

2 Hardware Environment

The results described in this paper 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.

3 The Effects of Sharing Resources

To gauge the performance effects of resources sharing, we run a set of single-processor benchmarks meant to stress each resource (main memory, I/O, and cpu). We then measure the slowdown incurred by each benchmark as we increase the number of its instances running concurrently on a single node.

The tests reveal that while a cpu-bound code does not slow down appreciably when multiple instances of it concurrently execute on a node, memory intensive codes suffer a slowdown between 10-60% and an I/O intensive code slows down over 1100%.

4 Mixing Jobs

To gauge how well alternate job mixes can improve performance, we repeat the experiments in Section 3, but utilize the unused processors in each experiment to concurrently execute other benchmarks that heavily utilize alternative resources. The results demonstrate that such benchmark mixes cause negligible slowdowns.

5 Symbiotic Space-Sharing and Parallel Codes

Having shown the extent to which resource contention exists and can be alleviated through improved processor allocation, we now ask how these findings can help speed parallel codes.

Parallel codes often employ every processor on each node in order to minimize occurrences of slower, inter-node communications. We test to see if the performance benefits of symbiotic space-sharing can outweigh these communication overheads.

We use runs of the NAS Parallel Benchmarks at 16 processors and execute each benchmark, first spread evenly across two 8-way nodes and then again spread evenly across four 8-way nodes. The results show that the performance of every benchmark (BT, MG, FT, LU, CG, IS, BTIO_EP, BTIO_Simple, BTIO_Full) increases by 12-55% when executing across four nodes instead of two.

We next show that this speedup can be maintained even when two benchmarks execute concurrently on the four nodes. We do this by pairing CPU, memory, and I/O bound codes. The speedup exhibited is between 3-38% for the selected benchmarks. We even attained speedup when mixing two memory-bound benchmarks which earlier tests had indicated might be symbiotic.

6 Prototype Symbiotic Scheduler

To test these concepts, we implemented a rudimentary symbiotic scheduler to compete against a standard EASY scheduler [2] meant to represent DataStar's policy.

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. The benchmarks were selected 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}. 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. For backfilling, each job is submitted with a synthetically generated expected runtime within 20% of the job's actual runtime. To constrain backfilling opportunities, at most twelve jobs occupy the queue at any given time.

The algorithm employed by the symbiotic scheduler is very simplistic. The scheduler partitions each node evenly into *top* and *bottom* halves and 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.

7 Conclusions

We have introduced symbiotic space-sharing as a promising technique for improving the performance efficiency of large-scale parallel machines. We have shown that a wide range of benchmarks commonly suffer between 10-60% slowdown due to memory resource contention and up to several orders of magnitude for I/O. We have shown that this effect can be mitigated by deploying alternate job mixes and have extended these results to parallel codes, demonstrating that node-sharing among parallel applications can increase throughput by increasing performance while maintaining high system utilization levels. We exhibited a prototype scheduler that improved throughput by 20%.

8 Acknowledgements

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.

References

- [1] <http://www.npaci.edu/DataStar/guide/home.html>.
- [2] 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.