THE UNIVERSITY OF CHICAGO

Decreasing End-to-End Job Execution Times by Increasing Resource Utilization using Predictive Scheduling in the Grid

Ioan Raicu

Computer Science Department

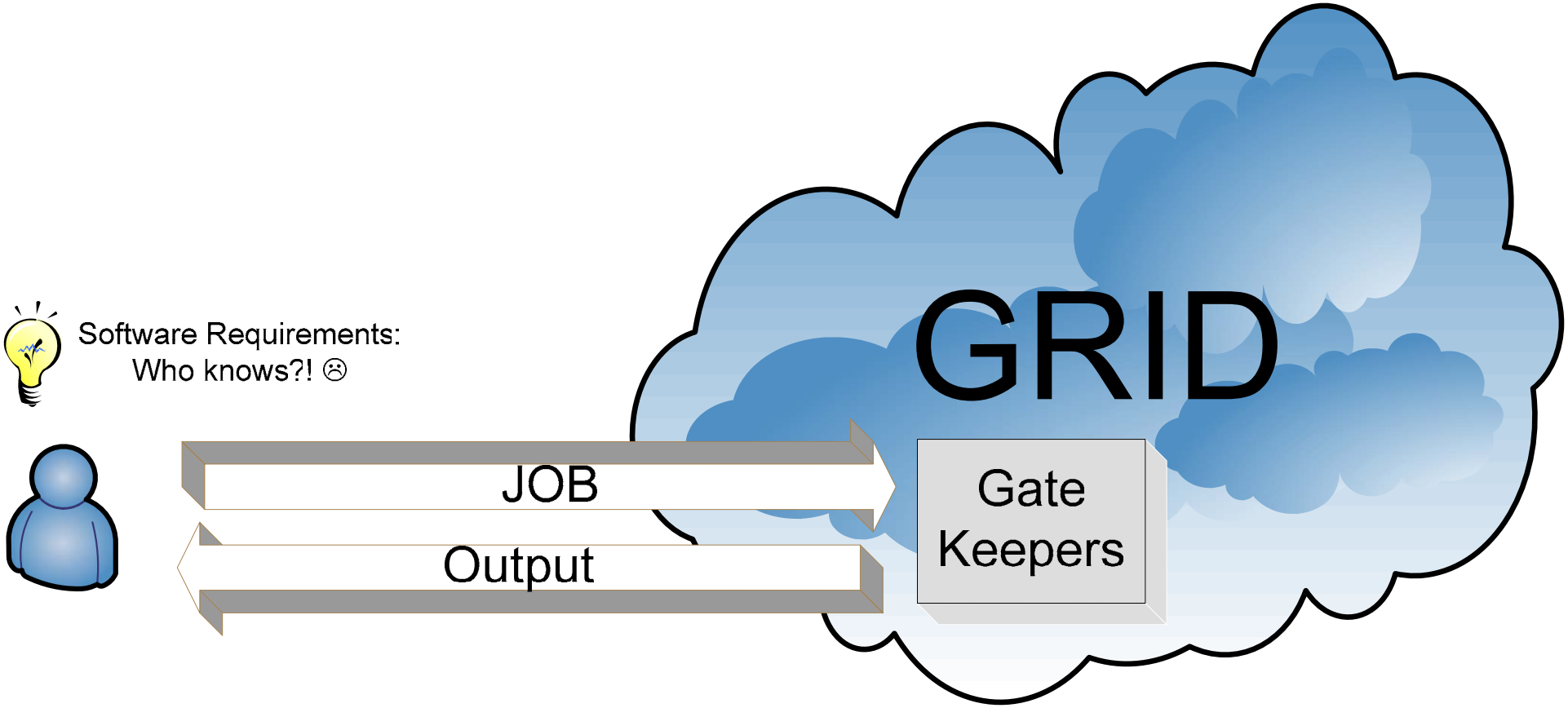
University of Chicago

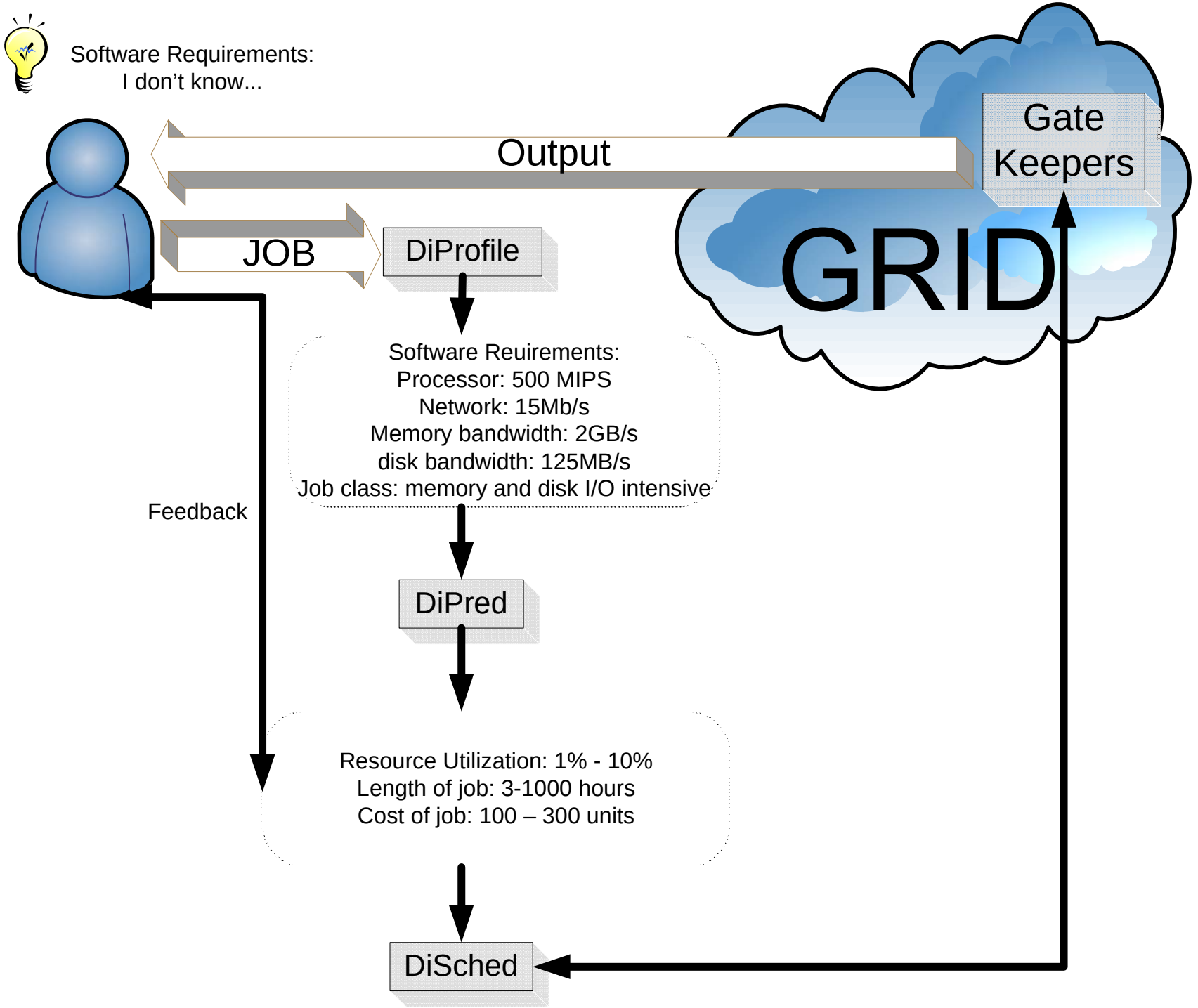
Grid Computing Seminar – Winter 2005

Motivation

- There is a gap between software requirements (high level) and hardware resources (low level)
  - Automatic mapping could produce better scheduling decisions and give users feedback with the expected running time of their software
  - **Can this automatic mapping be achieved for a wide range of software classes?**
- Large pool of resources in large scale distributed systems (i.e. Grid) are underutilized
  - Large (1000s of nodes) clusters or distributed systems seldom get more than a few percent resource utilization
  - Even modest improvements in resource utilization could have significant financial benefits
  - **Can software predictions aid scheduling decisions in resource management?**

Scenario


Software Requirements:
Who knows?! ☹
GRID
JOB
Output
Gate
Keepers


Software Requirements:
I don't know...
Output
Gate Keepers
GRID
JOB
DiProfile
Software Reuirements:
Processor: 500 MIPS
Network: 15Mb/s
Memory bandwidth: 2GB/s
disk bandwidth: 125MB/s
Job class: memory and disk I/O intensive
Feedback
DiPred
Resource Utilization: 1% - 10%
Length of job: 3-1000 hours
Cost of job: 100 – 300 units
DiSched

Related Work

- GridBench
- A Grid Resource Broker
- TITAN & PACE
- Prophesy & PAIDE
- Pegasus & Prophesy
- Application Development Software Project (GrADS)
- AppLeS (Application Level Scheduling)
- Flexible Co-Scheduling
- ClassAdd & RedLine

GridBench

- A set of tools to characterize Grid resources using micro-benchmarks
- GridBench framework runs benchmarks in Grid environments and collects, archives, and publishes the results
- **Differences, Limitations, and Comments:**
  - Uses benchmarks to gain insight into the performance of Grid resources
  - The relationship between the micro-benchmarks and the application kernel must be established; this relationship is found in a manual fashion that involves human intervention and detailed knowledge of the application or its empirical performance
- Papers
  - George Tsouloupas and Marios D. Dikaiakos. Characterization of Computational Grid Resources Using Low-level Benchmarks. Technical Report TR-2004-5, Dept. of Computer Science, University of Cyprus, December 2004
  - George Tsouloupas and Marios D. Dikaiakos. Gridbench: A tool for benchmarking grids. In Proceedings of the 4th International Workshop on Grid Computing (GRID2003), pages 60-67, Phoenix, AZ, November 2003.

A Grid Resource Broker

- Main features:
  - advance reservations
  - resource selection based on computer benchmark results and network performance predictions (using NWS)
  - basic adaptation facility
- **Differences, Limitations, and Comments:**
  - Benchmark-based procedure for resource selection
  - The user must identify relevant benchmarks and an estimated execution time on some specified resource
  - The authors assume linear scaling of the application in relation to the benchmark
- Papers:
  - Erik Elmroth and Johan Tordsson. A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection, PARA'04 State-of-the-Art in Scientific Computing, June 20-23, 2004.

TITAN & PACE

- TITAN: multi-tiered scheduling architecture
  - Focuses on the lowest tier which is responsible for local scheduling
- PACE: performance prediction system
  - Dynamically predict the execution time given an application model and suitable hardware descriptions
    - Resource tools are available to characterize the resource hardware through micro-benchmarking
    - Application tools are provided that take C source code and generate sub-tasks that capture the serial components of the code by control flow graphs
      - Straightforward for simple codes, and a library of templates exists for standard constructs
      - Applications that exhibit more complex parallel operations may require customization
- **Differences, Limitations, and Comments:**
  - Uses compiler technology to gain insight into application performance models
  - Applications models seem complex, and require user customization for non-trivial parallel operations
  - Requires access to source code, and the recompiling and re-linking of the source code
  - It is not clear how the control flow graph maps onto the micro-benchmarks in the resource selection problem
- Papers:
  - D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini, and G. R. Nudd. Local Grid Scheduling Techniques using Performance Prediction. IEE Proc. Comp. Digit. Tech., Nice, France, 150(2):87-96, April 2003.

Prophesy & PAIDE

- An infrastructure for performance analysis and modeling of parallel and distributed applications
  - automatic instrumentation of applications
    - Instruments the entire code via PAIDE at the level of loops and procedures
    - PAIDE includes a parser that identifies where to insert instrumentation code
  - databases for archival of information
    - Performance data from the performance database
    - Model templates from the template database
    - System characteristics from the systems database
  - automatic development of performance models using different techniques
    - Curve fitting (least squares)
    - Parameterized model of the application code
    - Coupling of the kernel models
- **Differences, Limitations, and Comments:**
  - Uses compiler technology to gain insight into application performance models
  - Performance models derived automatically (curve fitting) can only be used to explore an application's scalability
  - Performance models derived manually (parameterized models) can be used to explore what happens under different system configurations as well as different application sizes
  - Kernel coupling is not yet understood enough to be a useful technique
- Papers:
  - Valerie Taylor, Xingfu Wu, Rick Stevens. Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications, ACM SIGMETRICS Performance Evaluation Review, Volume 30, Issue 4, March 2003.

Pegasus & Prophesy

- A resource planner system consisting of:
  - The Pegasus workflow management and mapping system
    - Pegasus is used to map an abstract workflow description onto the available grid resources using Chimera or can be written directly by the user
    - Uses MDS to find available resources and characteristics
  - The Prophesy performance modeling infrastructure
    - Interfaces Pegasus and Prophesy to enable Pegasus to use the Prophesy prediction mechanisms
- **Differences, Limitations, and Comments:**
  - Pegasus provides a feasible solution, but it is not necessarily a low cost one in term of performance
  - General framework which does not take into consideration application specific performance characteristics
  - Not clear how Chimera is used to map an abstract workflow description onto the available grid resources
- Papers:
  - Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh and Scott Koranda, Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing, vol.1, no. 1, pages 25-39, 2003.
  - Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi and Miron Livny, Pegasus : Mapping Scientific Workflows onto the Grid , Across Grids Conference 2004, Nicosia, Cyprus, 2004.
  - Seung-Hye Jang, Xingfu Wu, Valerie Taylor, Gaurang Mehta, Karan Vahi, Ewa Deelman. Using Performance Prediction to Allocate Grid Resources. GriPhyN Technical Report 2004-25, July 2004.

Application Development Software Project (GrADS)

- GrADS goals: to realize a Grid system, by providing tools to manage all the stages of application development and execution through:
  - Problem solving environments
  - Grid compilers
    - Generates:
      - An intermediate representation code
      - Application performance models
  - Schedulers
    - Depends on the availability of two application-specific components:
      - performance model
        - » an analytic metric for the performance expected of the application on a given set of resources
      - Mapper
        - » provides directives for mapping logical application data or tasks to physical resources
  - Performance monitors
    - Used to offer performance guarantees and rescheduling
- **Differences, Limitations, and Comments:**
  - Uses compiler technology to gain insight into application performance models
  - It is unclear if the GrADS compiler has been realized where both the performance model and the mapper can be generated automatically
- Papers:
  - F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," International Journal of High Performance Computing Applications, vol. 15, pp. 327-344, 2001.
  - Holly Dail, Henri Casanova, and Fran Berman. A Decoupled Scheduling Approach for Grid Application Development Environments, Proceedings of Supercomputing, November 2002.

AppLeS (Application Level Scheduling)

- The AppLes approach exploits:
  - static and dynamic resource information
  - performance predictions (via NWS)
  - application and user-specific information
  - scheduling techniques that adapt “on-the-fly” to application execution
- **Differences, Limitations, and Comments:**
  - It requires to integrate in the application a scheduling agent which must be customized according to application features
  - The performance model is provided by the user
  - Typically uses an application-specific resource selection model to develop an ordered list of resource sets
- Papers:
  - F. Berman, R. Wolski, H. Casanova, et al. Adaptive Computing on the Grid Using AppLeS. IEEE Trans. on Parallel and Distrib. Systems, 14(5), 2003.

Flexible Co-Scheduling

- Flexible co-scheduling: Employs dynamic process classification and schedules processes using this class information
  - CS (co-scheduling): CS processes communicate often, and must be co-scheduled (gang-scheduled) across the machine to run effectively
  - F (frustrated): F processes have enough synchronization requirements to be co-scheduled, but due to load imbalance, they often cannot make full use of their allotted CPU time
  - DC (don't-care): DC processes rarely synchronize, and can be scheduled independently of each other without penalizing the system's utilization or the job's performance
  - RE (rate-equivalent): RE processes are characterized by jobs that have little synchronization, but require a similar (balanced) amount of CPU time for all their processes
- Processes are categorized based on measuring process statistics; this was achieved by implementing a lightweight monitoring layer that was integrated with MPI.
- Other Scheduling methods:
  - Gang scheduling: Combines time and space slicing: all processors are time-slices in a coordinated manner, and in each time slot, they are partitioned among multiple jobs
  - Backfilling: Improves the performance of pure space slicing by using small jobs from the end of the queue to fill in holes in the schedule
- **Differences, Limitations, and Comments:**
  - Established classes (CS, F, DC and RE) only allows the use of the appropriate scheduling decision, and does not aid in the mapping of processes and available resources
  - Applications need to run MPI and must be re-linked to the modified MPI library
- Papers:
  - Eitan Frachtenberg, Dror G. Feitelson, Juan Fernandez-Peinador, and Fabrizio Petrini. Parallel Job Scheduling under DynamicWorkloads. In 9th Workshop on Job Scheduling Strategies for Parallel Processing, June 2003.
  - Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini and Juan Fernandez. Adaptive Parallel Job Scheduling with Flexible CoScheduling. In IEEE Transactions on Parallel and Distributed Processing. To appear, 2005.
  - Eitan Frachtenberg, Dror G. Feitelson, Fabrizio Petrini, and Juan Fernandez. "Flexible CoScheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources", IPDPS 2003.

ClassAdd & RedLine

- A general-purpose resource selection framework to meet application requirements:
  - discovering resources
  - organizing resources
- ClassAdd & RedLine
  - Both application resource requirements and application performance models are specified declaratively
  - Mapping strategies can be determined by user-supplied code.
- **Differences, Limitations, and Comments:**
  - Focuses on the expressiveness of the declarative language and matchmaking problem
- Papers:
  - Chuang Liu, Lingyun Yang, Ian Foster, Dave Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications, HPDC-11, the Symposium on High Performance Distributed Computing, July 2002, Edinburgh, Scotland.
  - C. Liu and I. Foster. A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, 2003.

Same Goals…

- Mapping between software requirements and available resources
  - Benchmark resources
    - Requires user specified relationships between benchmarks and software requirements
  - Application performance models
    - Workflow analysis
      - It is hard and time consuming to produce accurate workflows
    - Uses compiler technology to gain insight into application performance models
      - Complex applications require user intervention
      - Performance models might not reflect actual performance on various architectures

… Different Approach

- Mapping between software requirements and available resources
  - Black box approach, generic and application independent
    - Build performance models to classify into several software classes based on the type and amount of resource usage
    - Considers interactions between various components in a system, and across distributed systems
    - No modifications needed to applications
- Lessons learned from related work
  - Could use benchmarking of Grid resources to enhance the resource selection
  - Could use co-scheduling based on the software classes to increase resource utilization
  - Could use historical information to recall the performance models generated for commonly used software
- Much work concentrated on lower levels of scheduling (i.e. local scheduling); it is important to address scheduling decisions on a larger global scale, giving hints to the low level schedulers

Proposed Work

- DiPerF
  - Collection of performance measurements in a distributed environment
- DiProfile
  - Perform software characterization
- DiPred
  - Automatic mapping of software requirements to raw resources
- DiSched
  - Matchmaking between the needed raw resources and the available resources

DiProfile Questions

1. Can the performance of a complex piece of software that heavily depends on its input be characterized in its entirety in a fraction of the time that it would take to run the entire workload?
2. Is the average case resource utilization sufficient to make the software class useful in real world resource management?
3. How flexible will the job profiler be to different types of software classes?
4. What is the maximum amount of time users are willing to wait for a job profile?
5. What overall performance improvement is needed in order to offset the extra complexity, time, and resources that the software profiler introduces?

DiPredict Questions

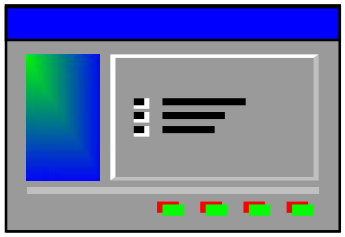
6. Can this automatic mapping (software requirements hardware resources) be achieved for a wide range of software classes?
7. Are the predictions accurate enough to be useful?
8. How flexible/fragile are the predictions?
9. Can the predictions be computed fast enough to satisfy the user agreeable wait time?

DiSched Questions

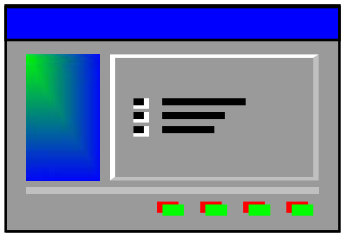
10. Can software performance predictions aid resource scheduling decisions in a consistent and significant manner?
11. Can a broad type of software (that identifies not resource usage, but rather software features, workload, usage patterns, etc) be defined that will most likely benefit from DiSched?
12. Are dynamic run-time performance models more accurate than static generic performance models, especially for certain types of software?

Co-Scheduling

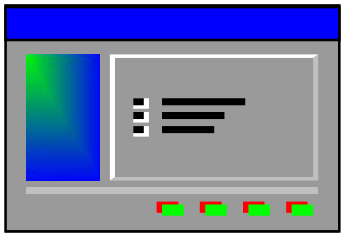
- Running several independent processes on 1 physical node
  - Allowing multiple independent processes to run will increase resource utilization, but it might also increase execution time
- Running several dependent processes on multiple physical nodes
  - Not allowing these dependent processes to run concurrently could negatively affect resource utilization and increase execution time
- Problem with Co-Scheduling: shared resources
  - Resource reservations becomes **VERY** hard
  - Resource planning is no longer an easy task



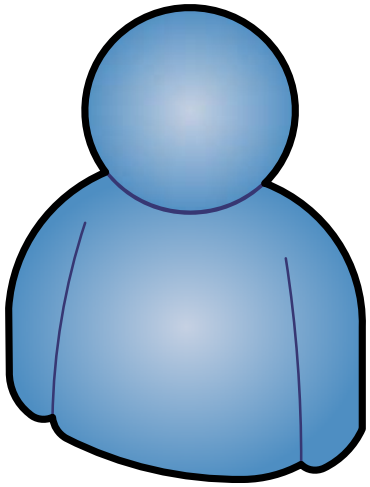
Software 1
Class: CPU
CPU: 100%
Network: 1%
Time: 100 sec



Software 2
Class: Network
CPU: 1%
Network: 100%
Time: 100 sec



Software 3
Class: Mixed
CPU: 50%
Network: 50%
Time: 100 sec



Time: 0 Software 1

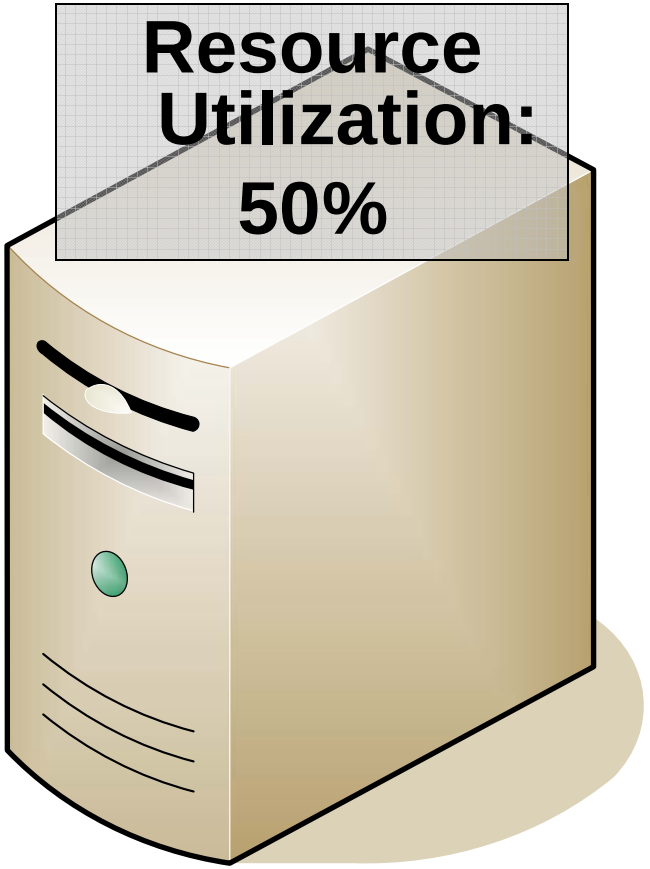
Time: 99 Output 1

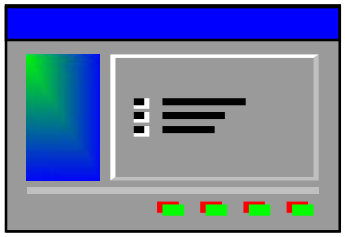
Time: 100 Software 2

Time: 199 Output 2

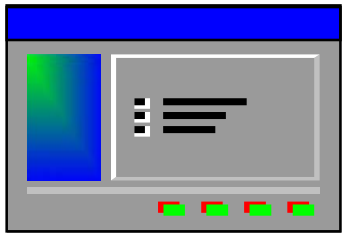
Time: 200 Software 3

Time: 299 Output 3

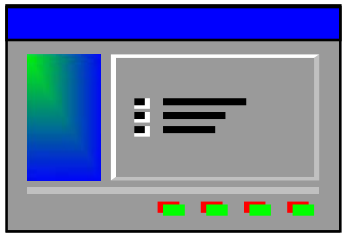




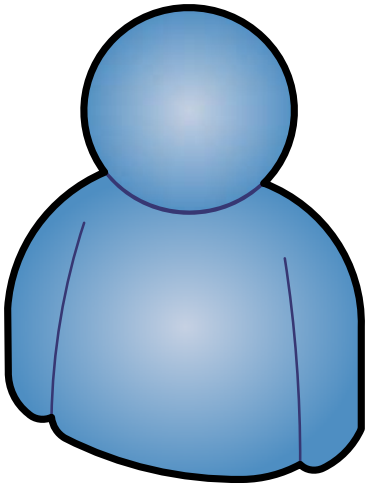
Software 1
Class: CPU
CPU: 100%
Network: 1%
Time: 100 sec



Software 2
Class: Network
CPU: 1%
Network: 100%
Time: 100 sec



Software 3
Class: Mixed
CPU: 50%
Network: 50%
Time: 100 sec



Time: 0 Software 1

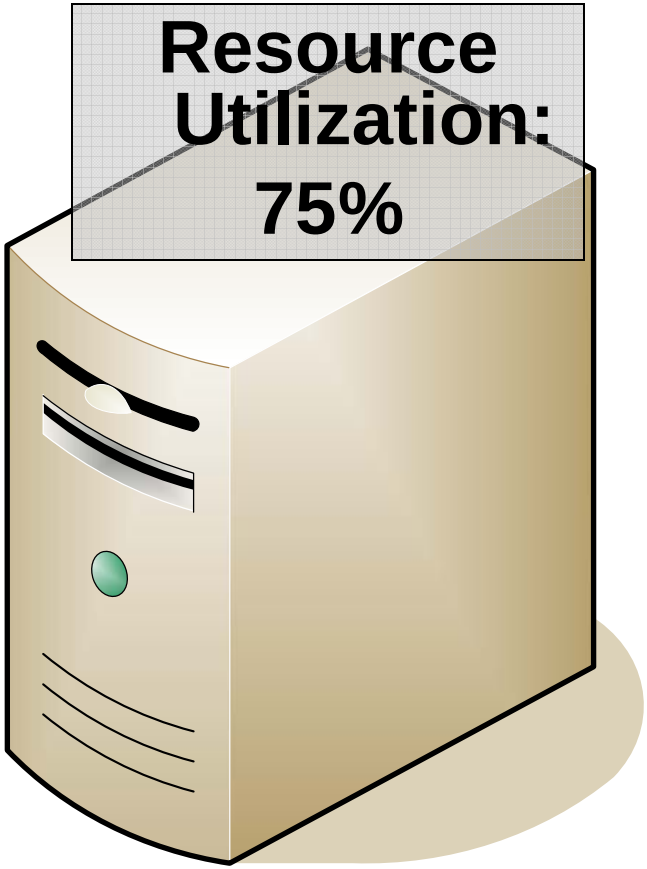
Time: 0 Software 2

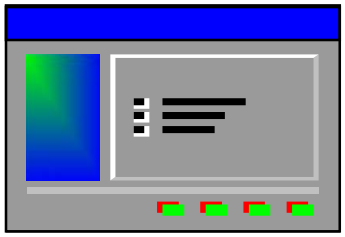
Time: 100 Output 1

Time: 100 Output 2

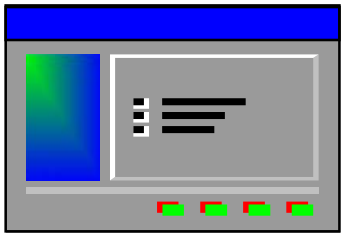
Time: 101 Software 3

Time: 200 Output 3

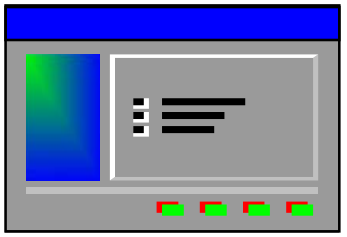




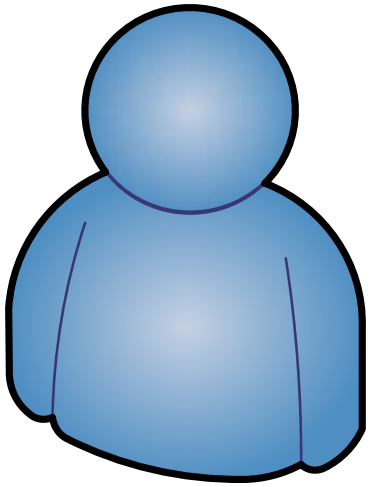
Software 1
Class: CPU
CPU: 100%
Network: 1%
Time: 100 sec



Software 2
Class: Network
CPU: 1%
Network: 100%
Time: 100 sec



Software 3
Class: Mixed
CPU: 50%
Network: 50%
Time: 100 sec



Time: 0 Software 1

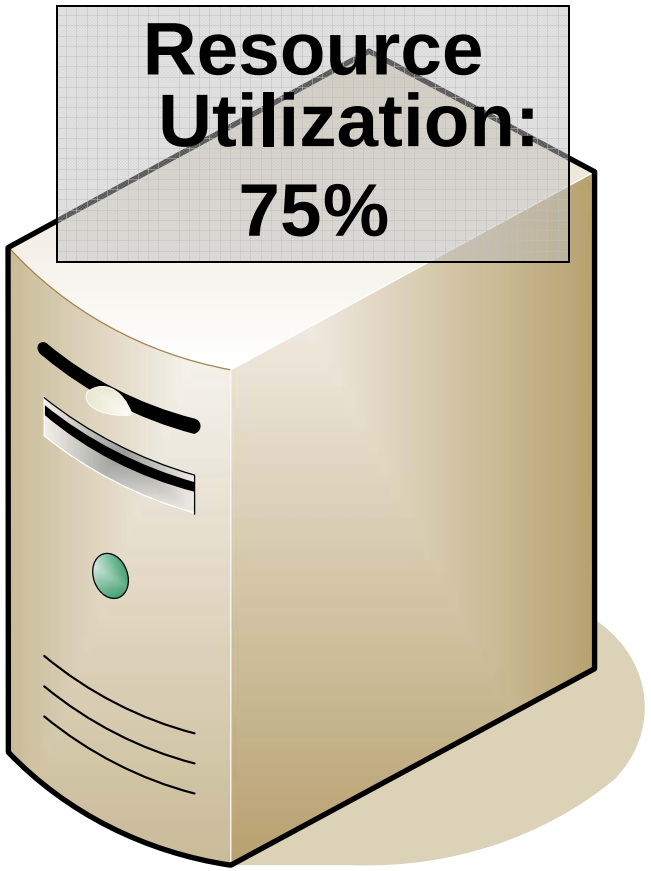
Time: 0 Software 2

Time: 0 Software 3

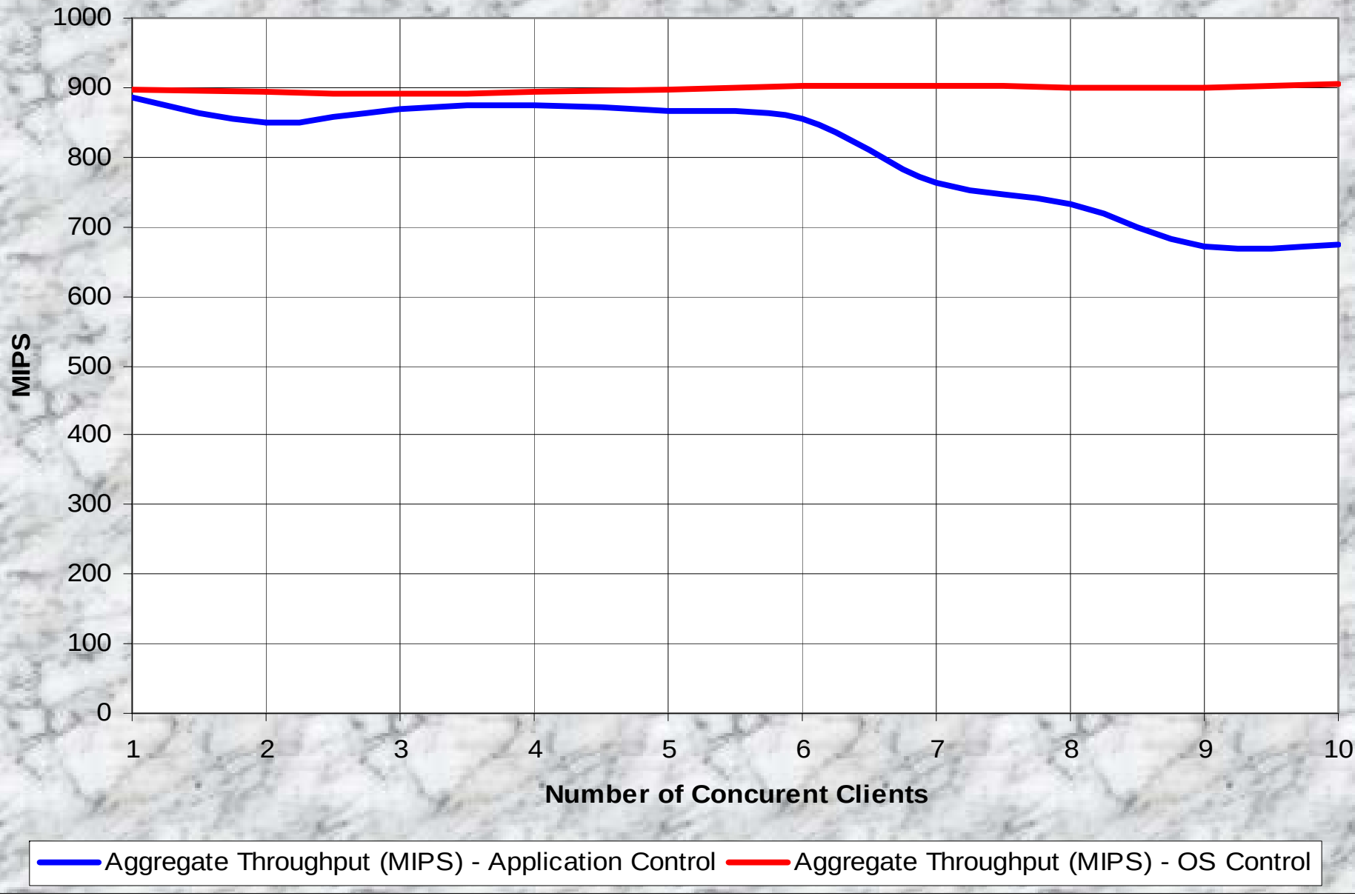
Time: 200 Output 1

Time: 200 Output 2

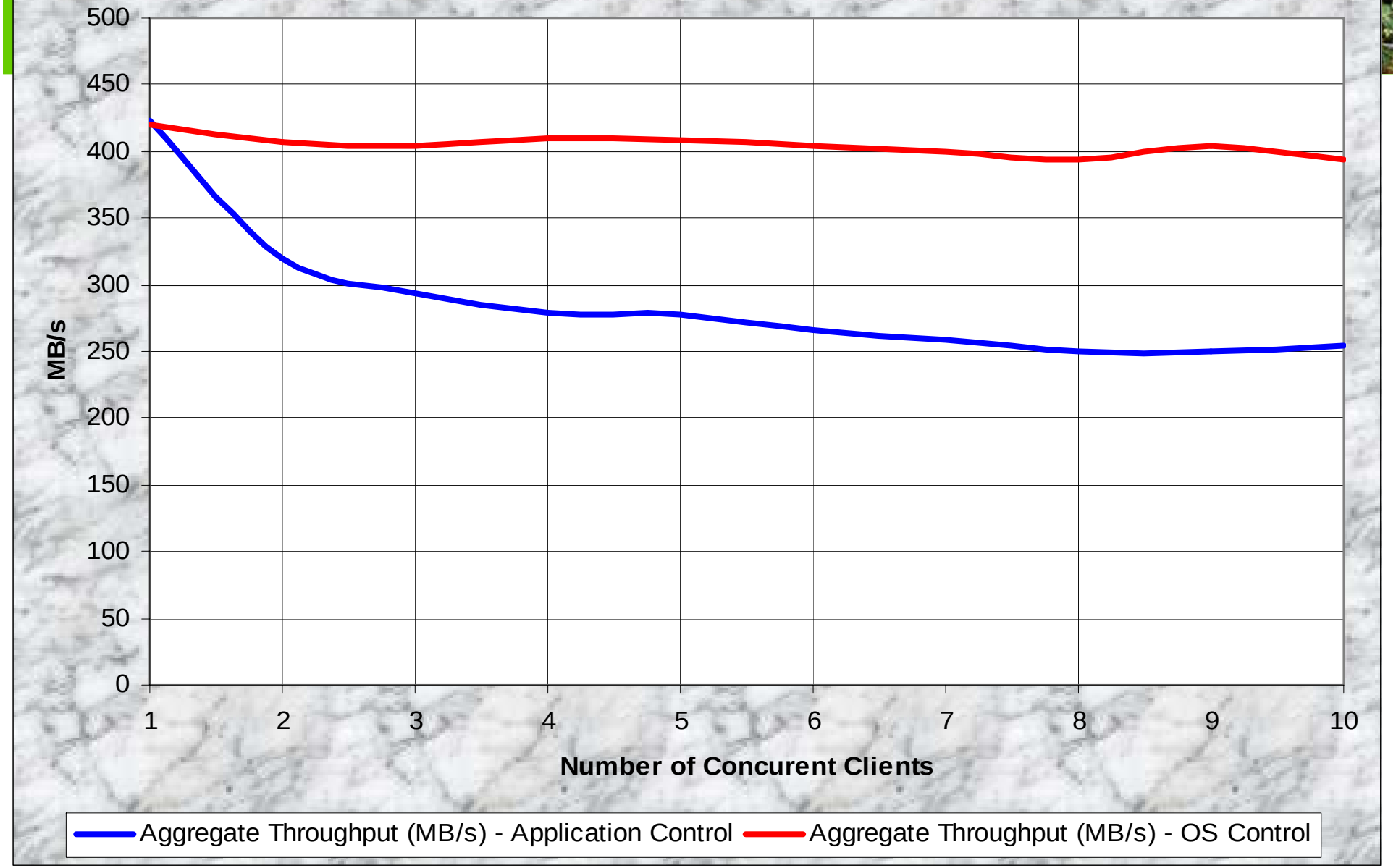
Time: 200 Output 3



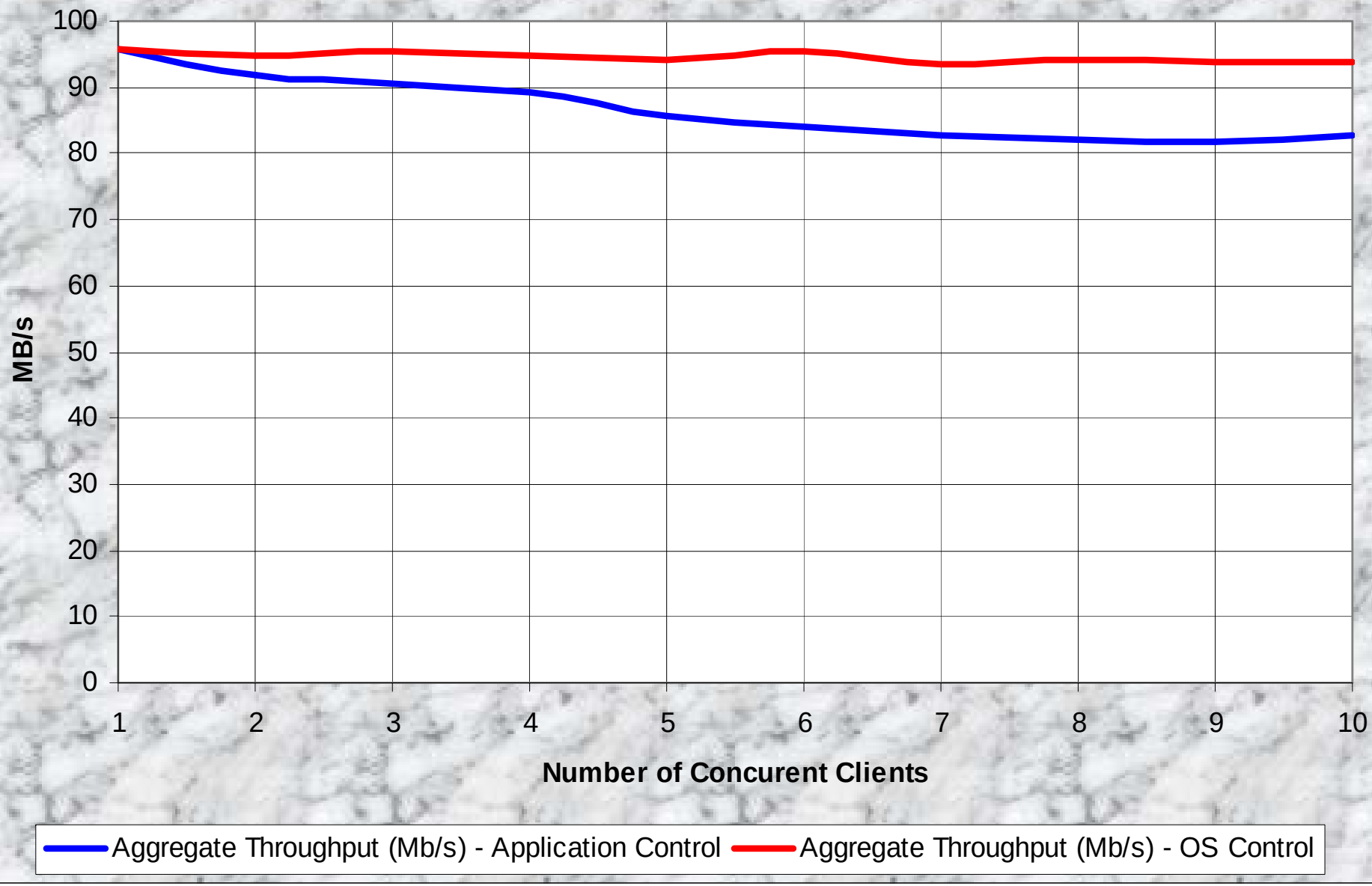
CPU


1000
900
800
700
600
500
400
300
200
100
0
MIPS
1
2
3
4
5
6
7
8
9
10
Number of Concurent Clients
Aggregate Throughput (MIPS) - Application Control
Aggregate Throughput (MIPS) - OS Control

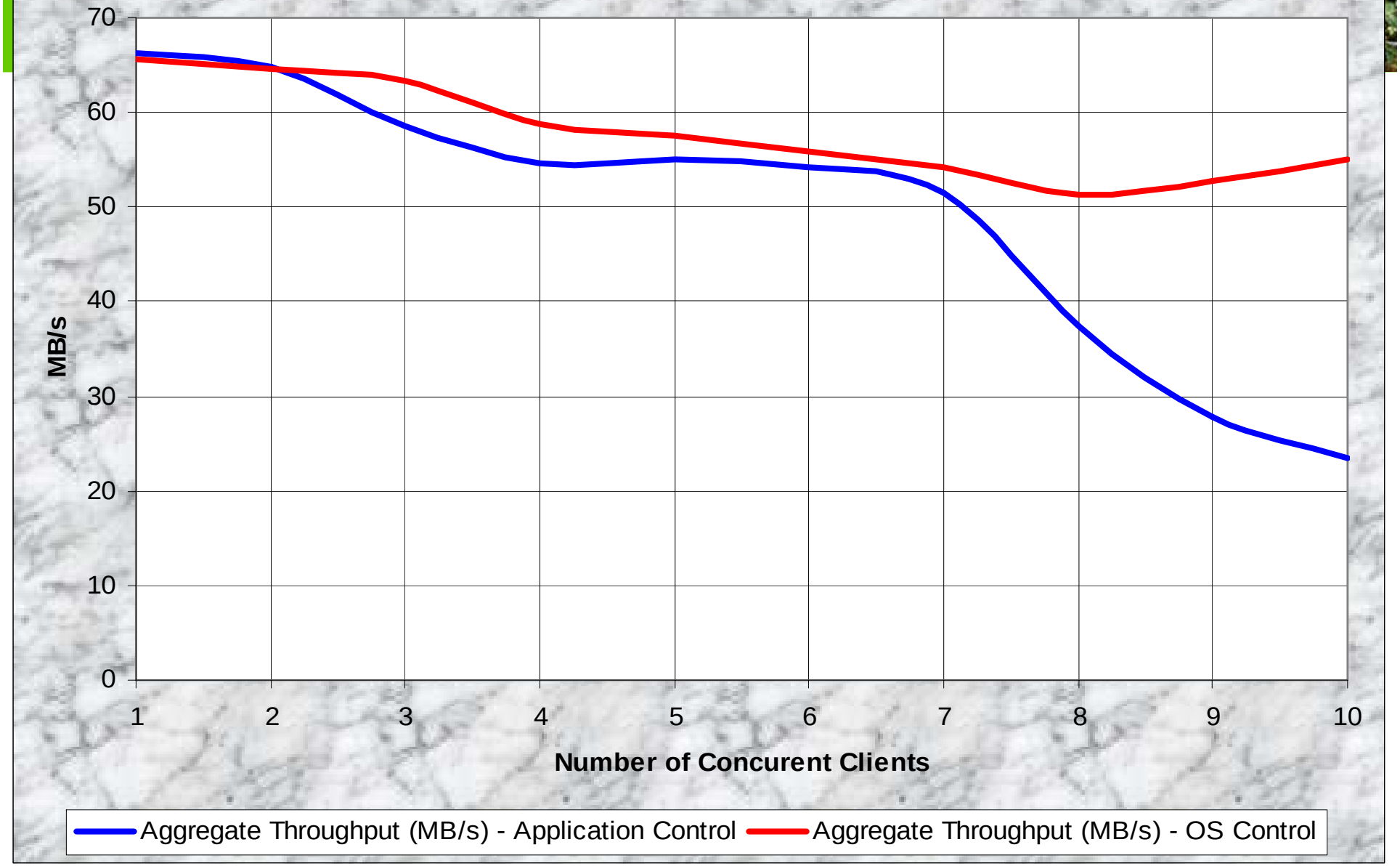
Memory


500
450
400
350
300
250
200
150
100
50
0
MB/s
1
2
3
4
5
6
7
8
9
10
Number of Concurent Clients
Aggregate Throughput (MB/s) - Application Control
Aggregate Throughput (MB/s) - OS Control

Network


100
90
80
70
60
50
40
30
20
10
0
MB/s
1
2
3
4
5
6
7
8
9
10
Number of Concurent Clients
Aggregate Throughput (Mb/s) - Application Control
Aggregate Throughput (Mb/s) - OS Control

Disk


70
60
50
40
30
20
10
0
MB/s
1
2
3
4
5
6
7
8
9
10
Number of Concurent Clients
Aggregate Throughput (MB/s) - Application Control
Aggregate Throughput (MB/s) - OS Control

Co-Scheduling: Test Case 1

Overview		
	CPU %	**Performance**
mem (MB/s)	100%	423 MB/s
disk (MB/s)	12%	66 MB/s
cpu (MIPS)	100%	886 MIPS
nic (Mb/s)	14%	95 Mb/s

Test Case 1					
	Capacity	**CPU %**	**Parallel**	**Serial**	**% Perf**
mem (MB/s)	1%	1%	4.1	4.2	98%
disk (MB/s)	1%	< 1%	0.6	0.7	94%
cpu (MIPS)	95%	95%	825.9	842.3	98%
nic (Mb/s)	1%	< 1%	0.9	1.0	95%
Total CPU %		96%	**Overall Performance**		96%

Co-Scheduling: Test Case 2

Test Case 2					
	Capacity	**CPU %**	**Parallel**	**Serial**	**% Perf**
mem (MB/s)	95%	95%	389.9	402.4	97%
disk (MB/s)	1%	< 1%	0.6	0.7	93%
cpu (MIPS)	1%	1%	8.8	8.9	99%
nic (Mb/s)	1%	< 1%	0.9	1.0	98%
Total CPU %		96%	**Overall Performance**		97%

Co-Scheduling: Test Case 3

Test Case 3					
	Capacity	**CPU %**	**Parallel**	**Serial**	**% Perf**
mem (MB/s)	1%	1%	4.2	4.2	98%
disk (MB/s)	100%	12%	65.6	66.3	99%
cpu (MIPS)	1%	1%	8.8	8.9	99%
nic (Mb/s)	100%	14%	93.8	95.8	98%
Total CPU %		28%	**Overall Performance**		99%

Co-Scheduling: Test Case 4

Test Case 4					
	Capacity	**CPU %**	**Parallel**	**Serial**	**% Perf**
mem (MB/s)	20%	20%	79.812	84.7	94%
disk (MB/s)	100%	12%	57.121	66.3	86%
cpu (MIPS)	61%	61%	410.8	540.8	76%
nic (Mb/s)	50%	7%	43.1	47.9	90%
Total CPU %		100%	**Overall Performance**		87%

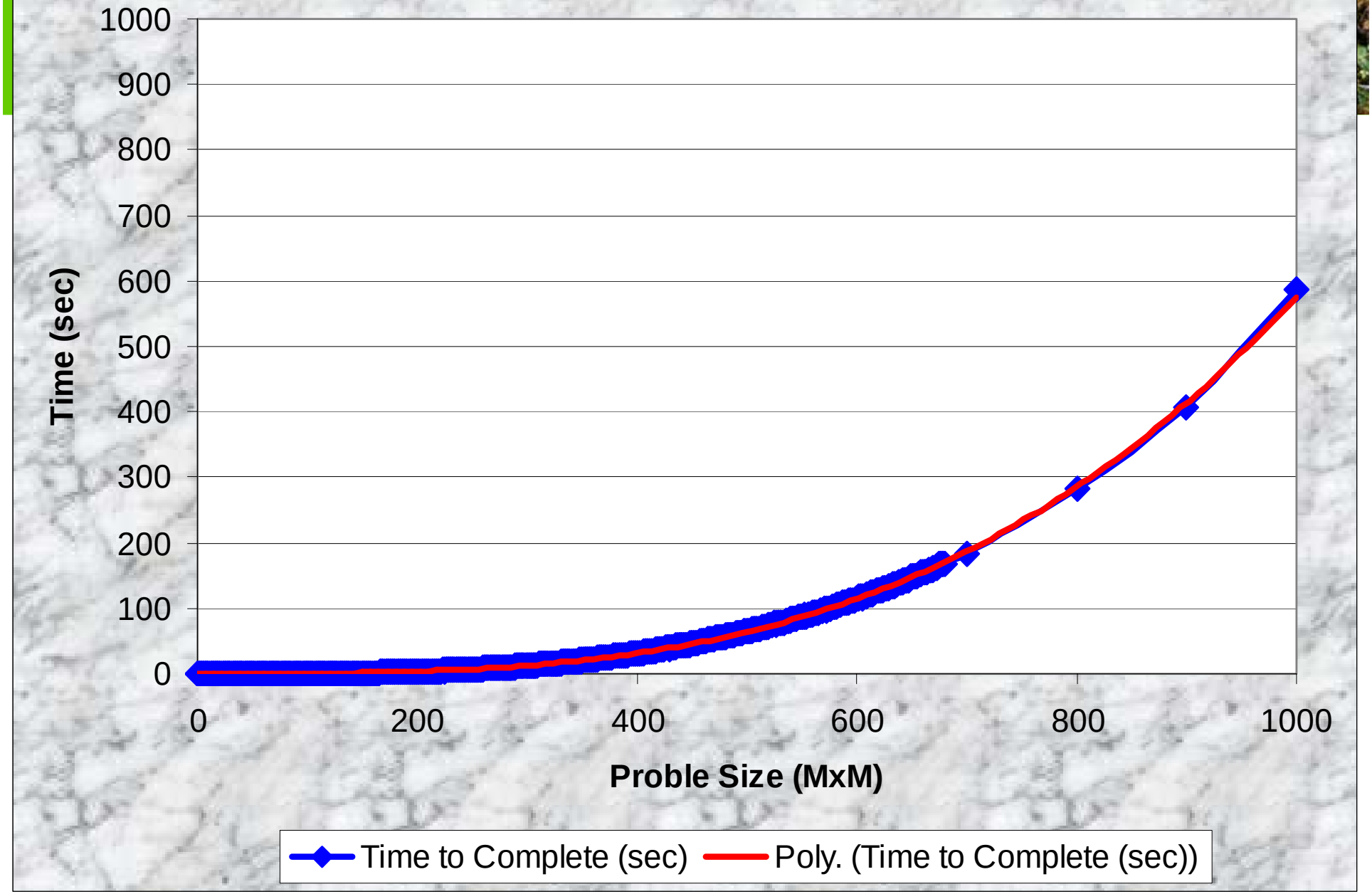
Co-Scheduling: Test Case 5

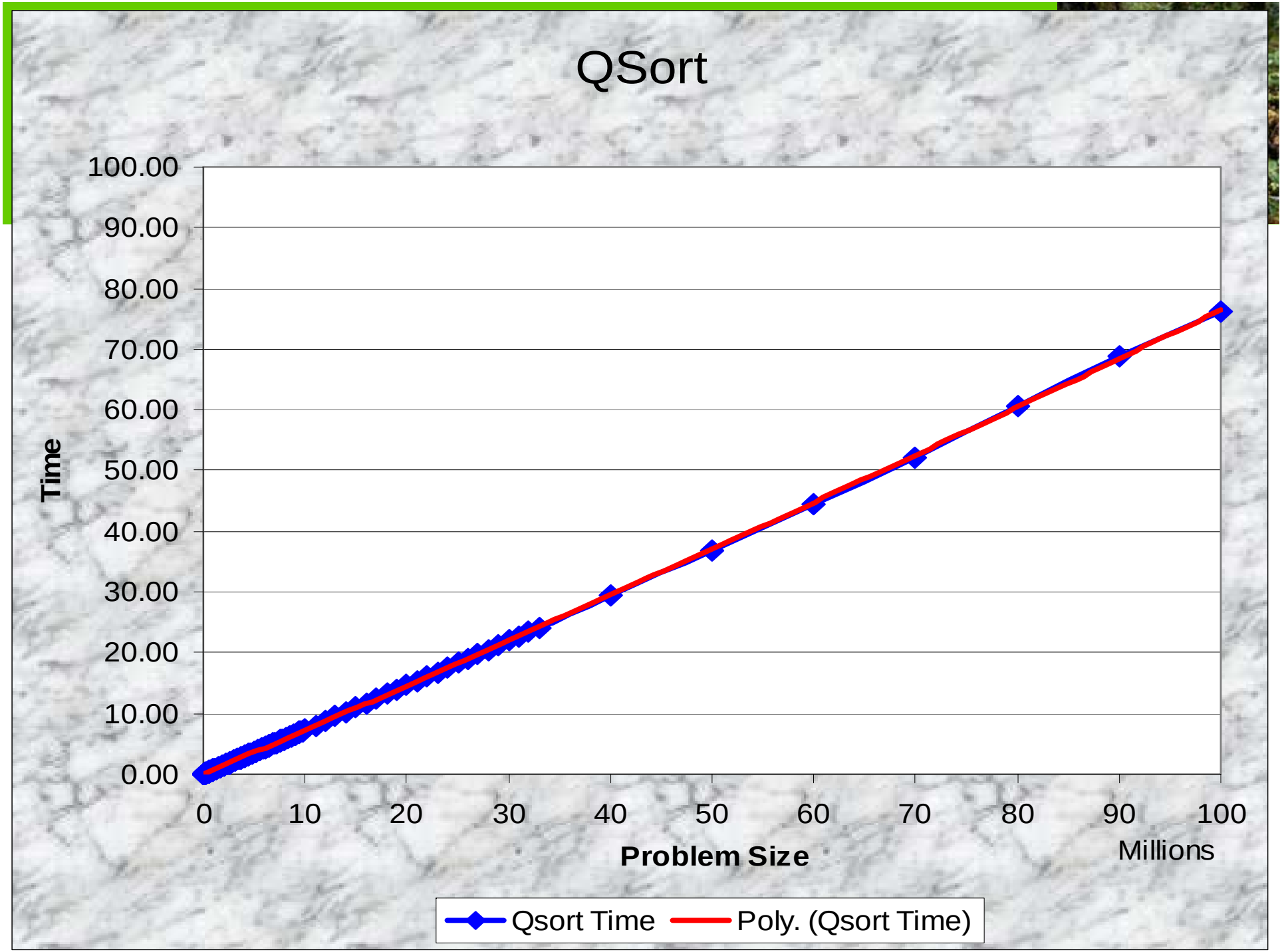
Test Case 5					
	Capacity	**CPU %**	**Parallel**	**Serial**	**% Perf**
mem (MB/s)	37%	37%	139.3892	156.7	89%
disk (MB/s)	100%	12%	57.6732	66.3	87%
cpu (MIPS)	37%	37%	250.8	328.1	76%
nic (Mb/s)	100%	14%	84.2	95.8	88%
Total CPU %		100%	**Overall Performance**		85%

Software Characterization

- Problems
  - Jacobi 2D ~ $O(N^2)$
    - Solving a linear system of equations
    - Regular iterative code continuously updates a 2D matrix within a loop body between global error-checking stages
    - Utilizes CPU intensively
    - Problem size: from 1x1 to 1000x1000 input space
  - Quick Sort ~ O(N) + O(N*log(N))
    - Sorts a list of integers
    - Utilizes mostly hard disk, and some CPU
    - Problem size: from 1 to 100000000 integer input space ~ 1GB data

Jacobi 2D


1000
900
800
700
600
500
400
300
200
100
0
Time (sec)
0
200
400
600
800
1000
Proble Size (MxM)
Time to Complete (sec)
Poly. (Time to Complete (sec))


QSort
100.00
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
Time
0
10
20
30
40
50
60
70
80
90
100
Millions
Problem Size
Qsort Time
Poly. (Qsort Time)

Conclusion

- Completed
  - An implementation of DiPerF
  - A literature survey
  - Showed that co-scheduling is possible for several different classes of software without significant loss of performance
  - Showed that reliable software performance characterization is possible with only a fraction of the entire input space
  - A proposal outlining the entire system of components and the related work
  - Showed that reliable software performance characterization is possible with only a fraction of the entire input space (at least for 2 specific problems – jacobi and qsort)
- Future work
  - Implement the entire system and show that predictive scheduling can decreasing end-to-end job execution times and increasing resource utilization

Questions?

- More info:
  - http://people.cs.uchicago.edu/~iraicu/research/uchicago/cs33340-05/
- Questions?

THE UNIVERSITY OF CHICAGO