Using Small Abstractions to Program Large Distributed Systems

Douglas Thain
University of Notre Dame
19 February 2009

Using Small Abstractions to Program Large Distributed Systems

(And multicore computers!)

Douglas Thain
University of Notre Dame
19 February 2009

Clusters, clouds, and grids
give us access to zillions of CPUs.

How do we write *programs* that can
run effectively in large systems?

I have 10,000 iris images acquired in my research lab. I want to reduce each one to a feature space, and then compare all of them to each other. I want to spend my time doing science, not struggling with computers.

I have a laptop.

I own a few machines

I can buy time from Amazon or TeraGrid.

Now What?

How do a I program a CPU?

- I write the algorithm in a language that I find convenient: C, Fortran, Python, etc…
- The compiler chooses instructions for the CPU, even if I don’t know assembly.
- The operating system allocates memory, moves data between disk and memory, and manages the cache.
- To move to a different CPU, recompile or use a VM, but ***don’t change the program***.

How do I program the grid/cloud?

- Split the workload into pieces.
  - *How much work to put in a single job?*
- Decide how to move data.
  - *Demand paging, streaming, file transfer?*
- Express the problem in a workflow language or programming environment.
  - *DAG / MPI / Pegasus / Taverna / Swift ?*
- Babysit the problem as it runs.
  - *Worry about disk / network / failures…*

How do I program on 128 cores?

- Split the workload into pieces.
  - *How much work to put in a single thread?*
- Decide how to move data.
  - *Shared memory, message passing, streaming?*
- Express the problem in a workflow language or programming environment.
  - *OpenMP, MPI, PThreads, Cilk, …*
- Babysit the problem as it runs.
  - *Implement application level checkpoints.*

Tomorrow's distributed systems will be clouds of multicore computers.

Can we solve ***both*** problems with a single model?

Observation

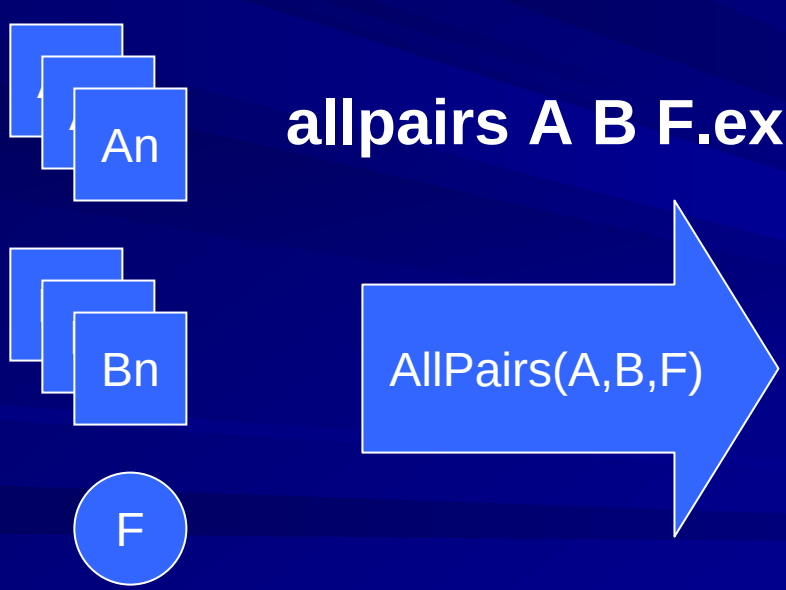
- In a given field of study, a single person may repeat the same ***pattern*** of work many times, making slight changes to the data and algorithms.
- Examples everyone knows:
  - Parameter sweep on a simulation code.
  - BLAST search across multiple databases.
- Are there other examples?.

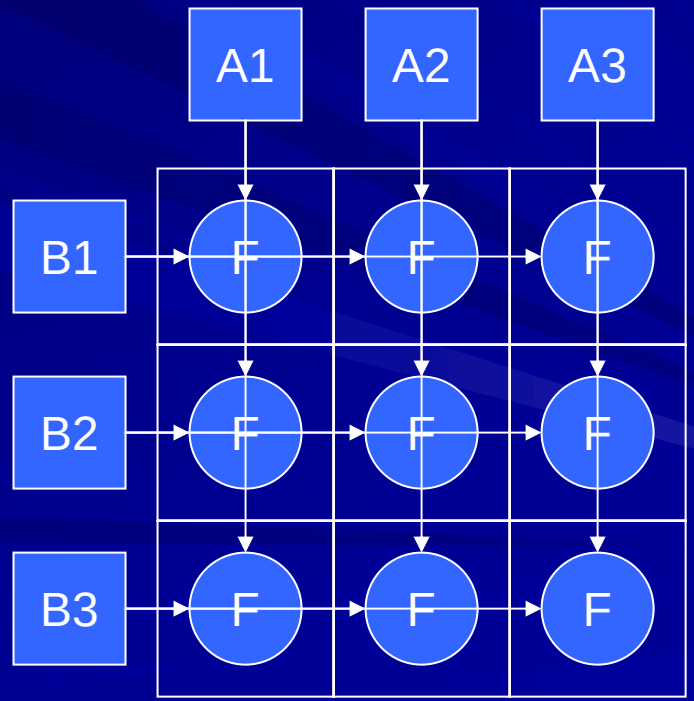
Abstractions for Distributed Computing

- Abstraction: a ***declarative specification*** of the computation and data of a workload.
- A ***restricted pattern***, not meant to be a general purpose programming language.
- Uses ***data structures*** instead of files.
- Provide users with a ***bright path***.
- Regular structure makes it tractable to model and ***predict performance.***

All-Pairs Abstraction

AllPairs(set A, set B, function F)
returns matrix M where
M[i][j] = F(A[i], B[j]) for all i,j



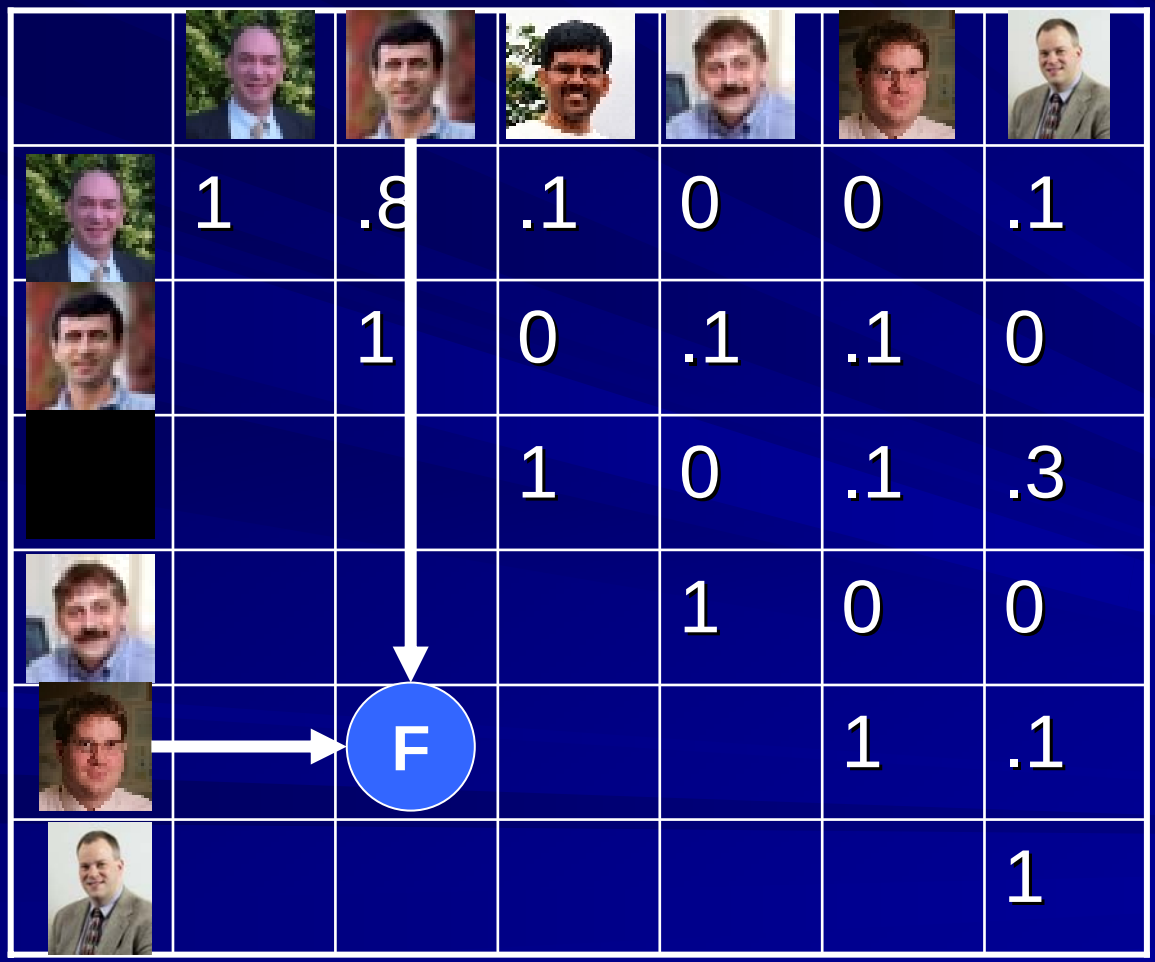


Moretti, Bulosan, Flynn, Thain,
AllPairs: An Abstraction… IPDPS 2008

Example Application

- **Goal: Design robust face comparison function.**

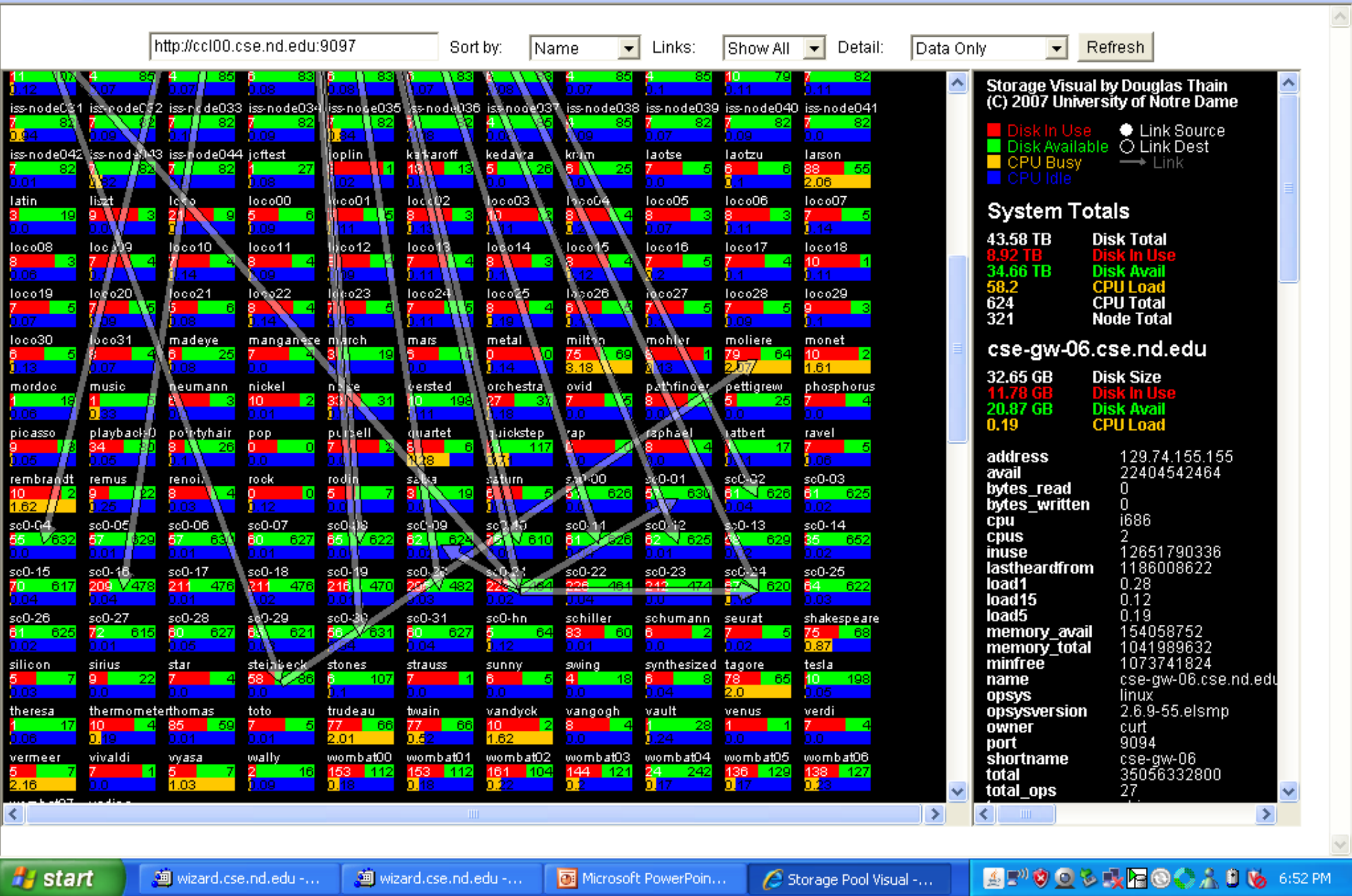
Similarity Matrix Construction



Current Workload:
4000 images
256 KB each
10s per F
(five days)

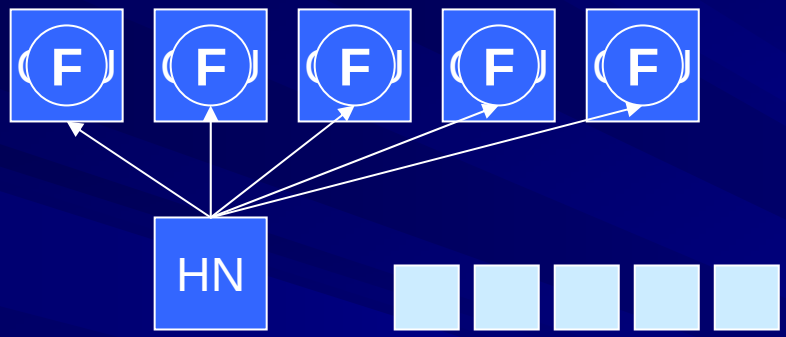
Future Workload:
60000 images
1MB each
1s per F
(three months)

http://www.cse.nd.edu/~ccl/viz

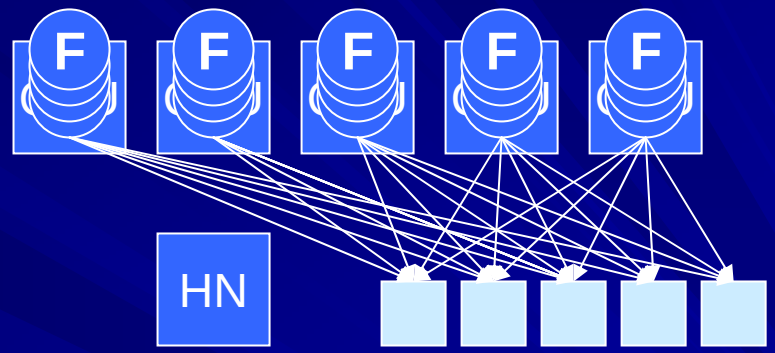


Non-Expert User Using 500 CPUs

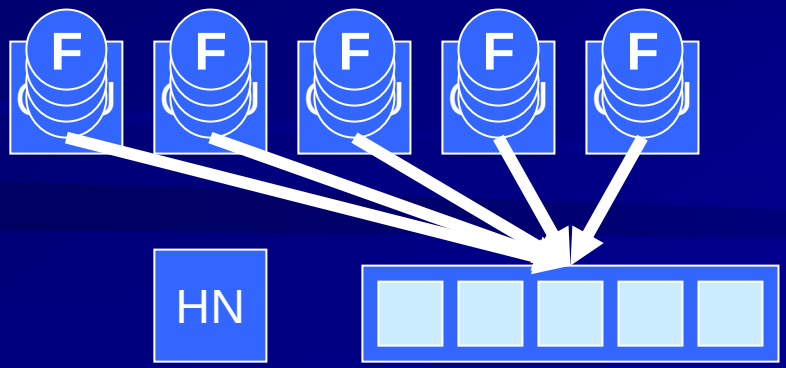
Try 1: Each F is a batch job.
Failure: Dispatch latency >> F runtime.



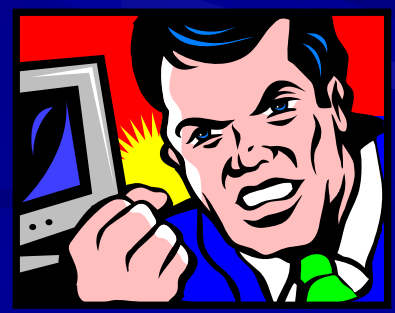
Try 2: Each row is a batch job.
Failure: Too many small ops on FS.



Try 3: Bundle all files into one package.
Failure: Everyone loads 1GB at once.

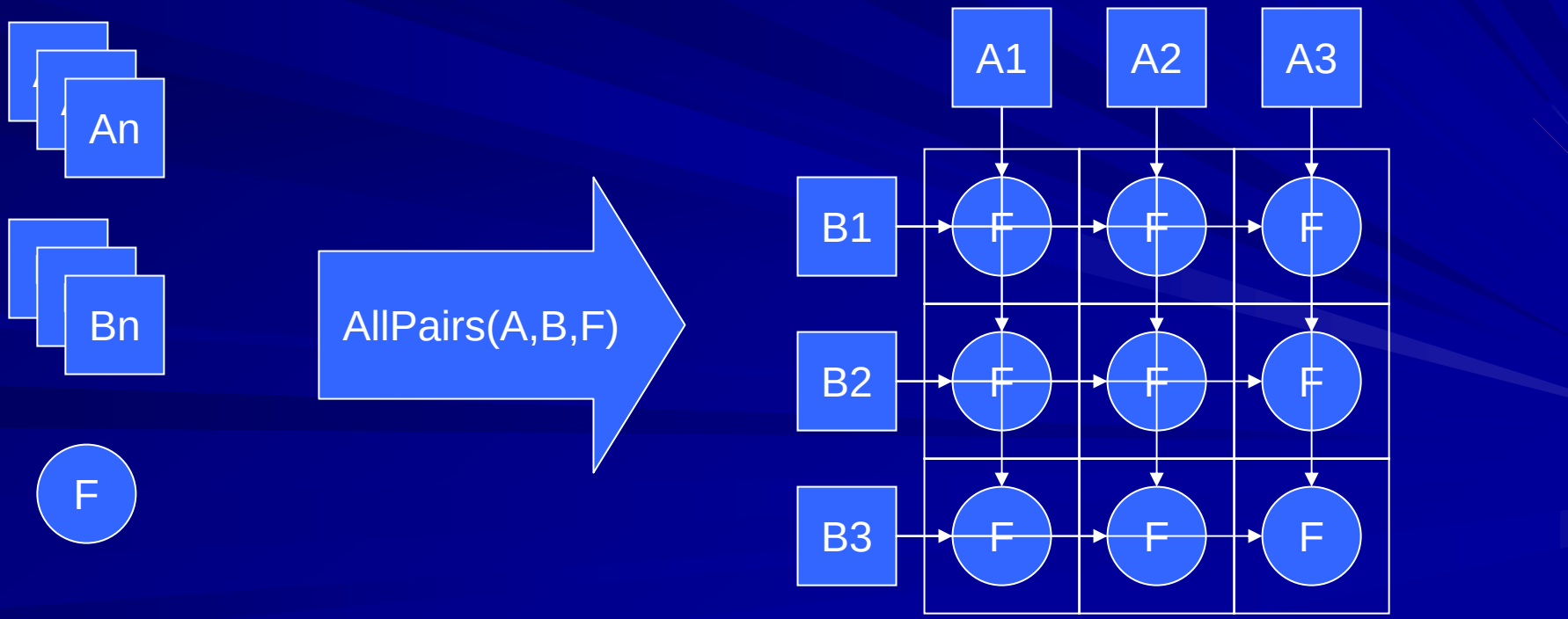


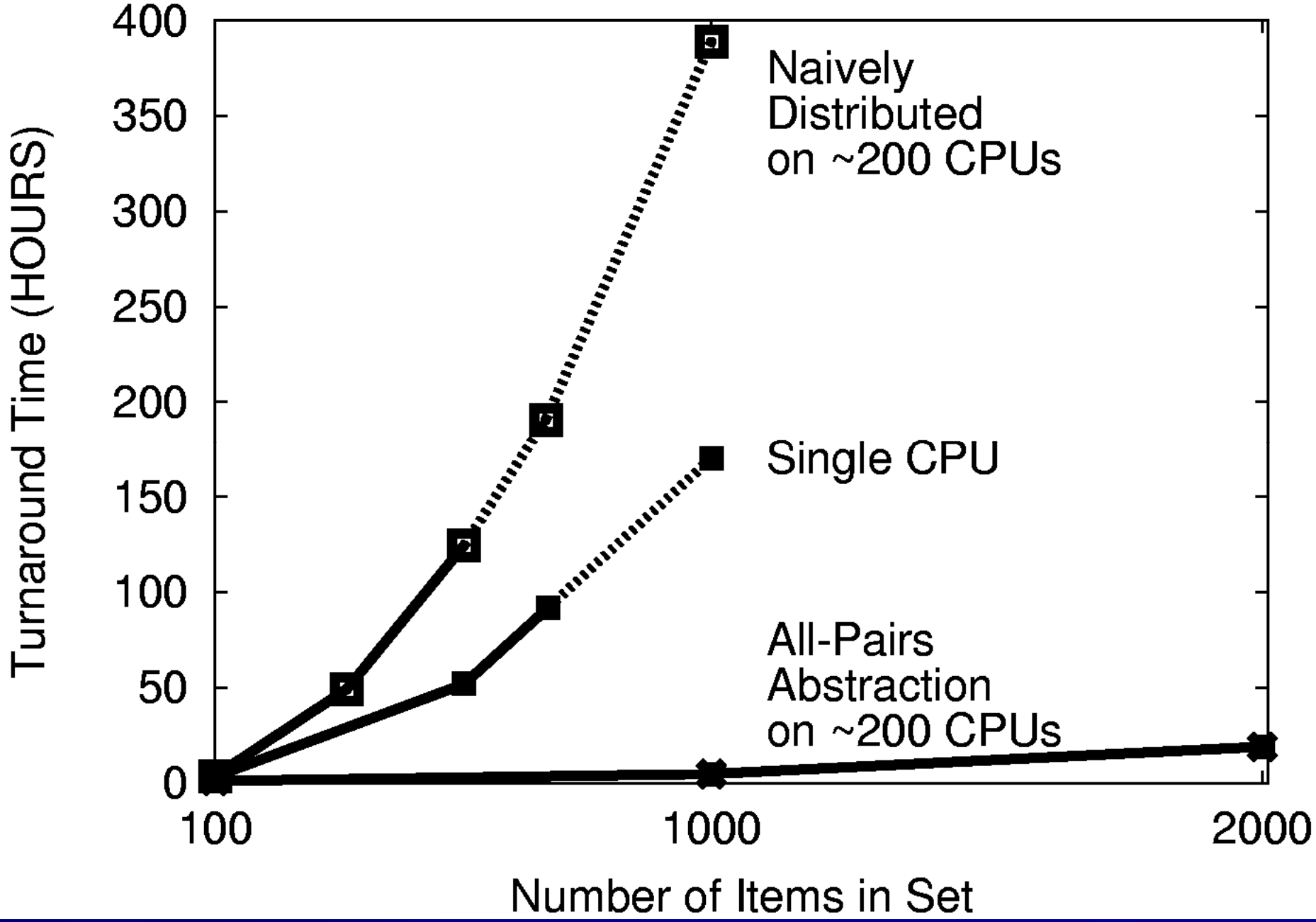
Try 4: User gives up and attempts to solve an easier or smaller problem.



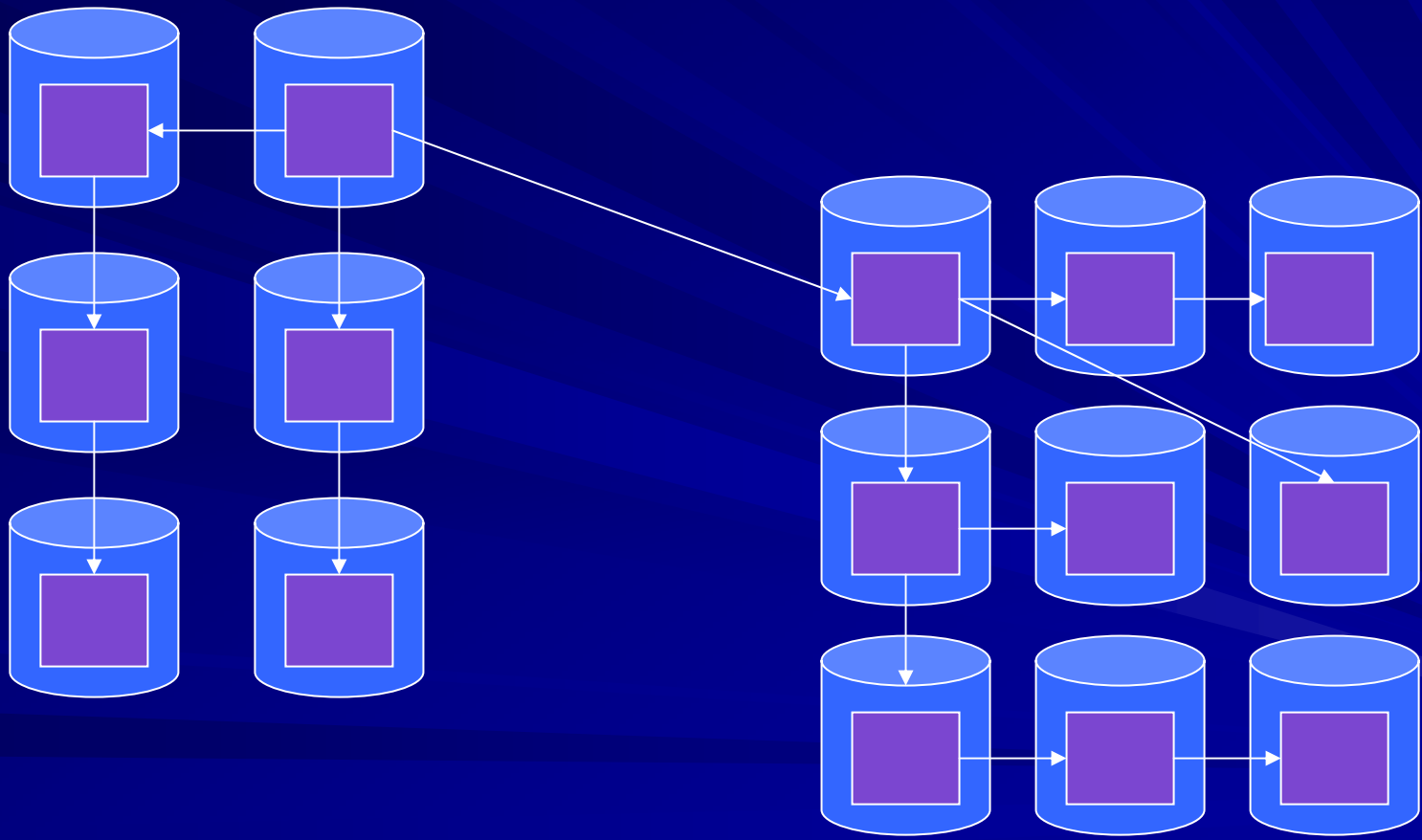
All-Pairs Abstraction

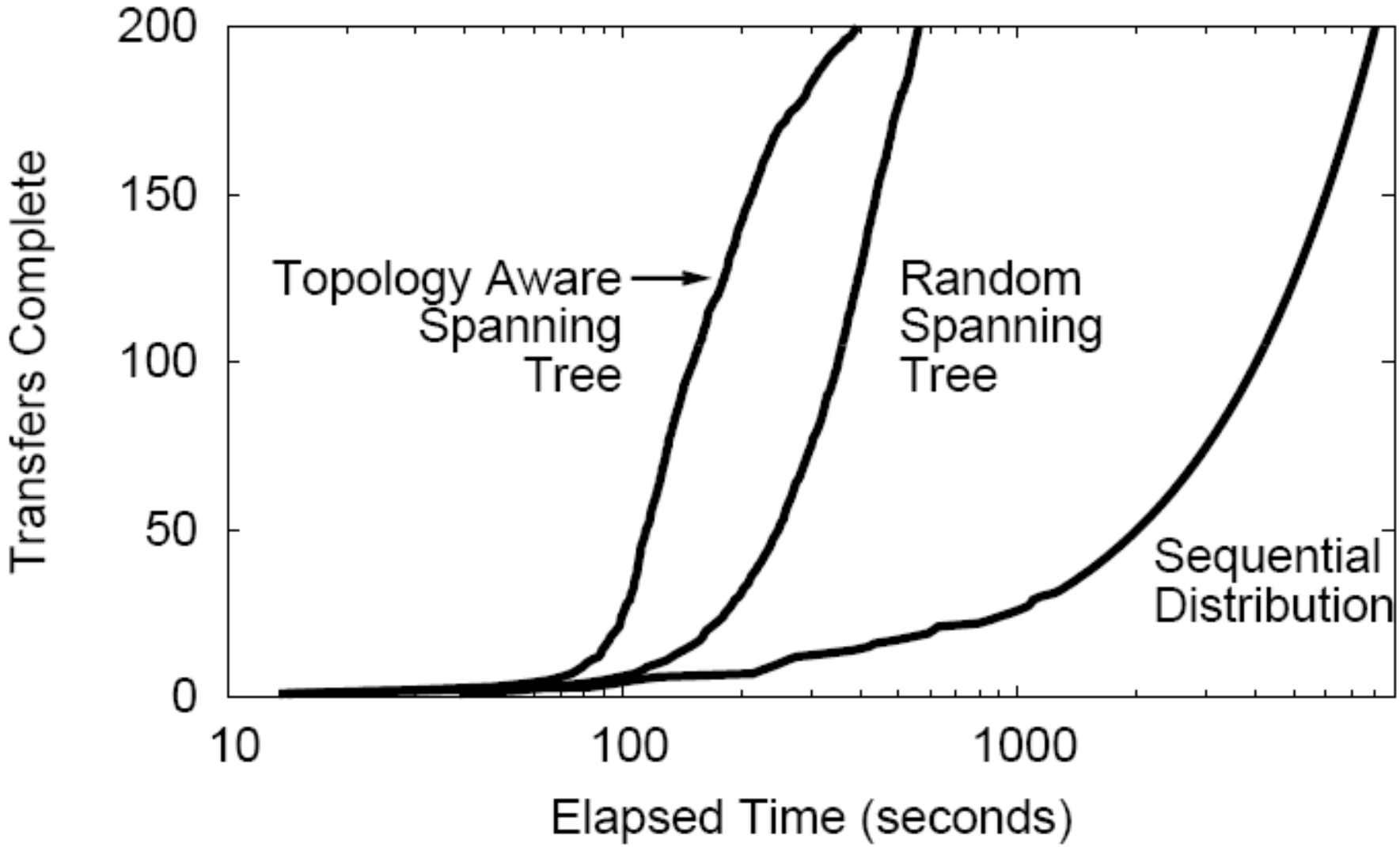
AllPairs(set A, set B, function F)
returns matrix M where
M[i][j] = F(A[i], B[j]) for all i,j




400
350
300
250
200
150
100
50
0
Turnaround Time (HOURS)
100
1000
2000
Number of Items in Set
Naively
Distributed
on ~200 CPUs
Single CPU
All-Pairs
Abstraction
on ~200 CPUs

Distribute Data Via Spanning Tree

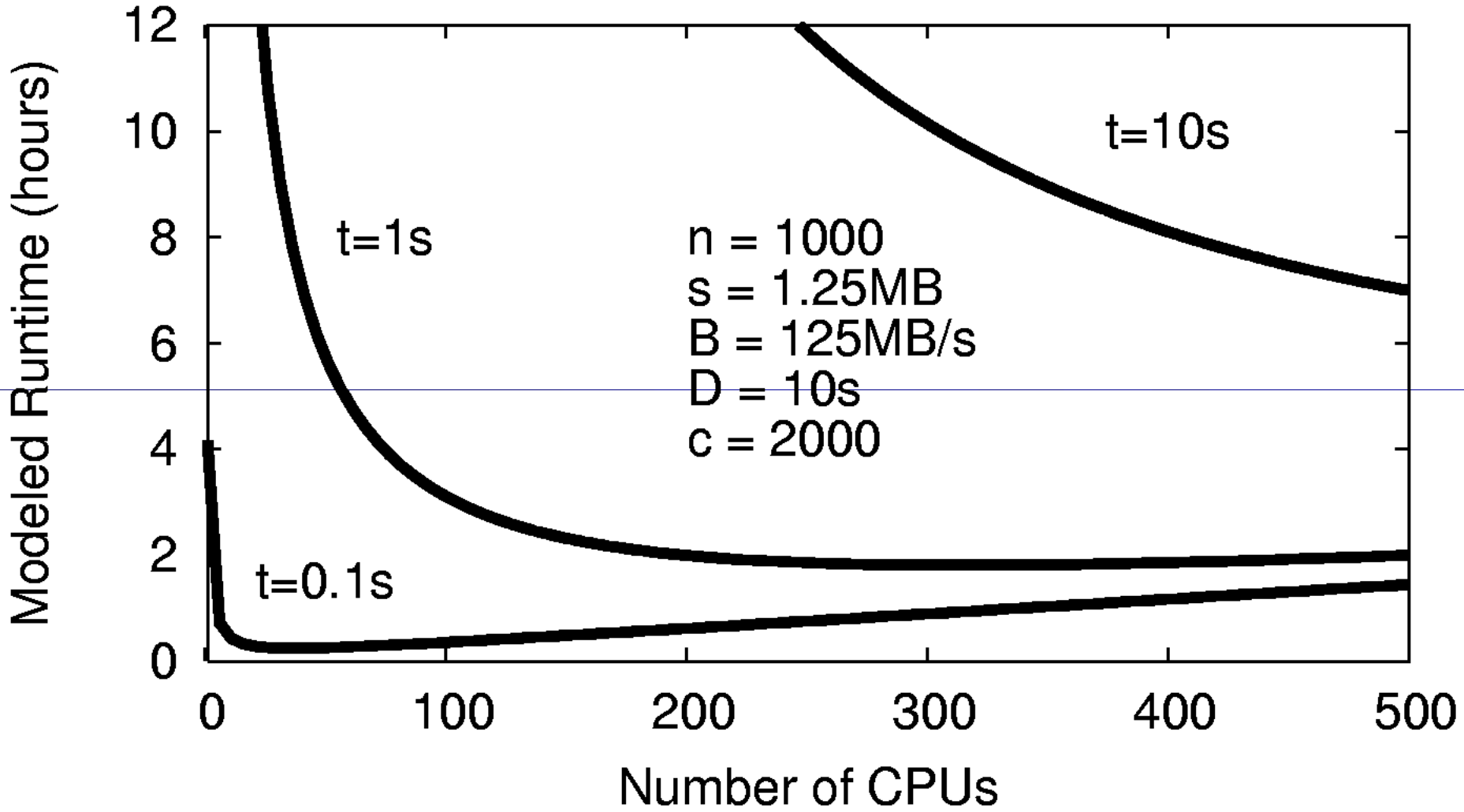



200
150
100
50
0
Transfers Complete
10
100
1000
Elapsed Time (seconds)
Topology Aware
Spanning
Tree
Random
Spanning
Tree
Sequential
Distribution

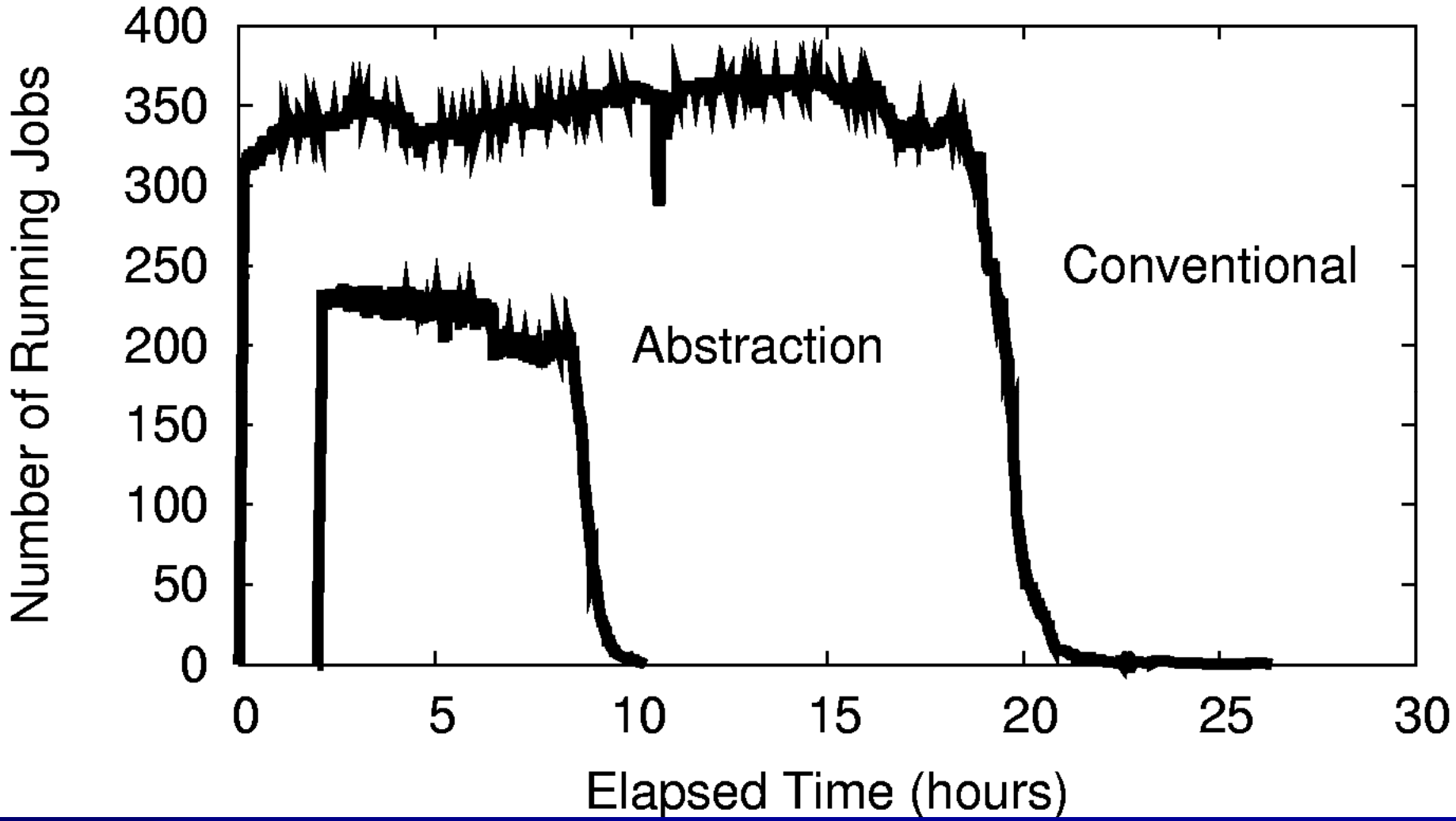
An Interesting Twist

- Send the absolute minimum amount of data needed to each of N nodes from a central server
  - Each job must run on exactly 1 node.
  - Data distribution time: **O(D sqrt(N))**
- Send all data to all N nodes via spanning tree distribution:
  - Any job can run on any node.
  - Data distribution time: **O(D log(N))**
- **It is both faster and more robust to send all data to all nodes via spanning tree.**

Choose the Right # of CPUs



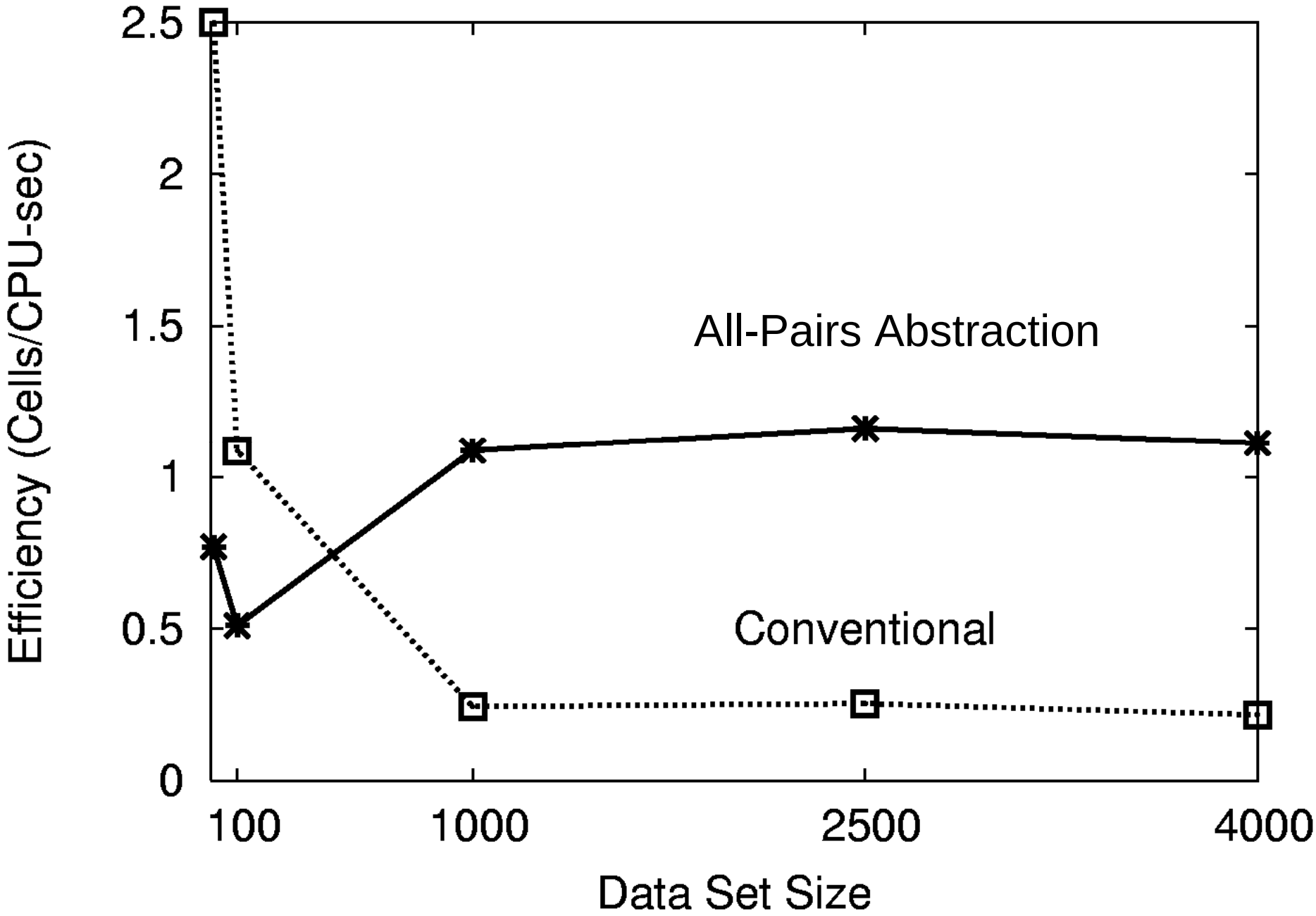
What is the right metric?



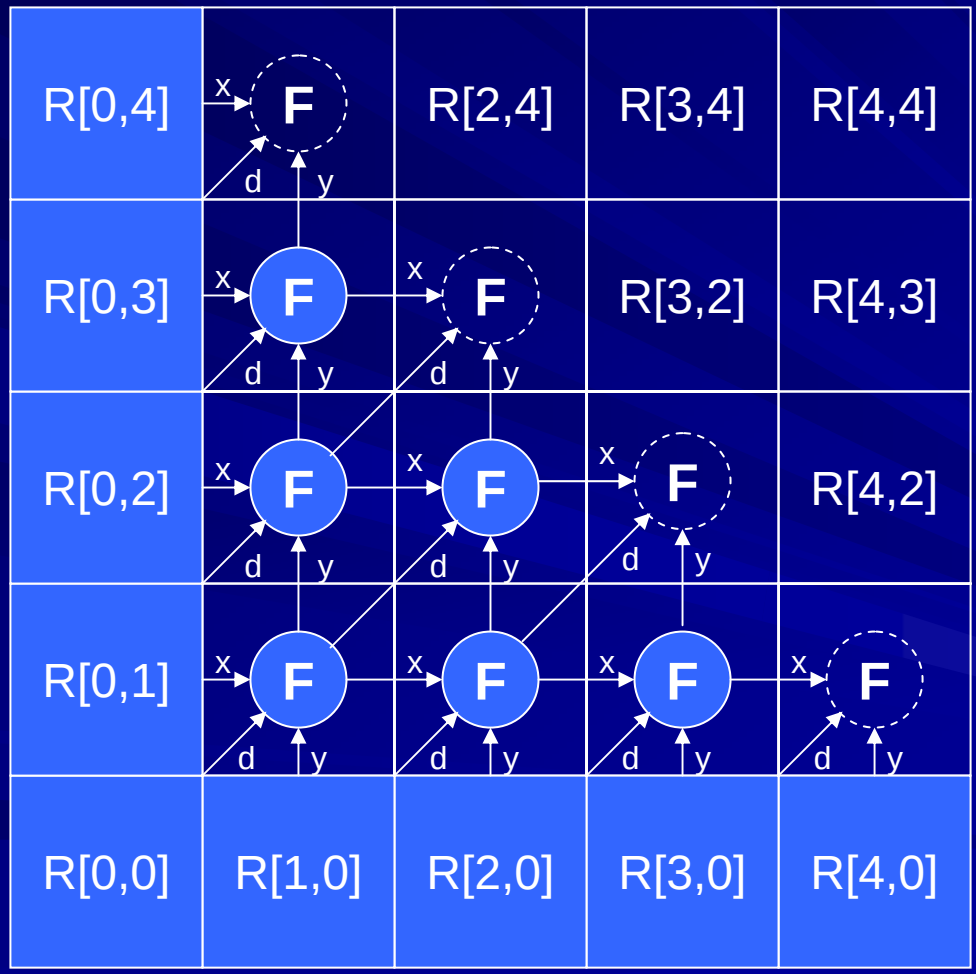
How to measure in clouds?

- Speedup?
  - Seq Runtime / Parallel Runtime
- Parallel Efficiency?
  - Speedup / N CPUs?
- Neither works, because the number of CPUs varies over time and between runs.

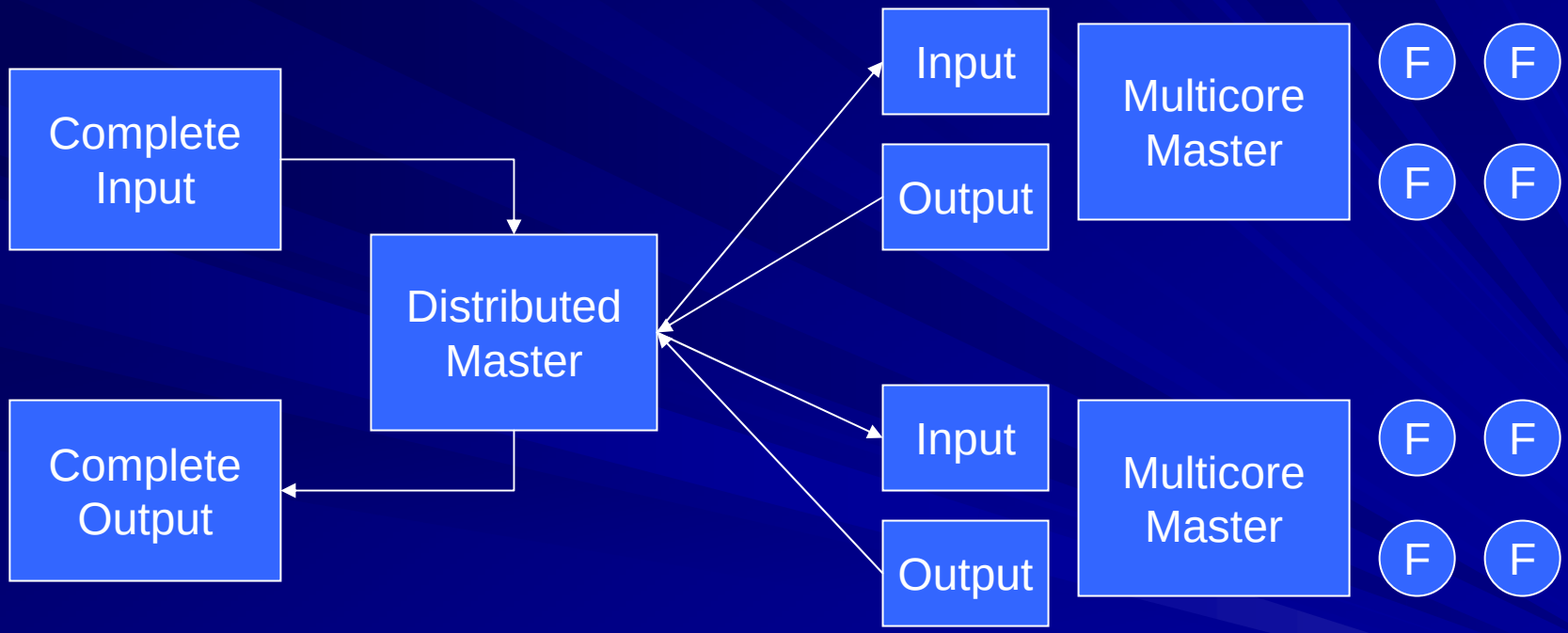
- ***An Alternative: Cost Efficiency***
  - Work Completed / Resources Consumed
  - Cars: Miles / Gallon
  - Planes: Person-Miles / Gallon
  - Results / CPU-hours
  - Results / $$$


Efficiency (Cells/CPU-sec)
2.5
2
1.5
1
0.5
0
All-Pairs Abstraction
Conventional
100
1000
2500
4000
Data Set Size

Wavefront (R[x,0], R[0,y], F(x,y,d))



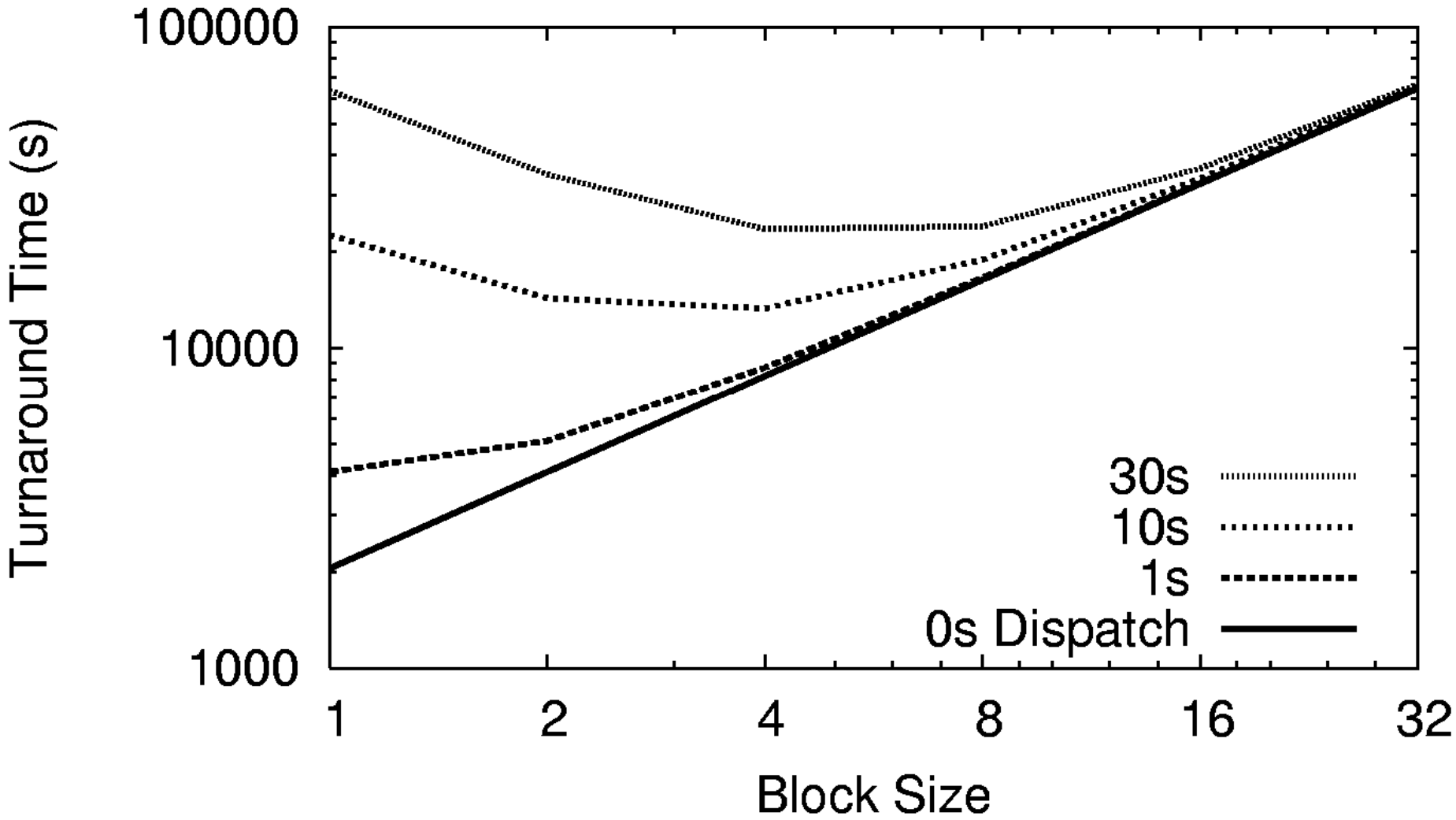
Implementing Wavefront



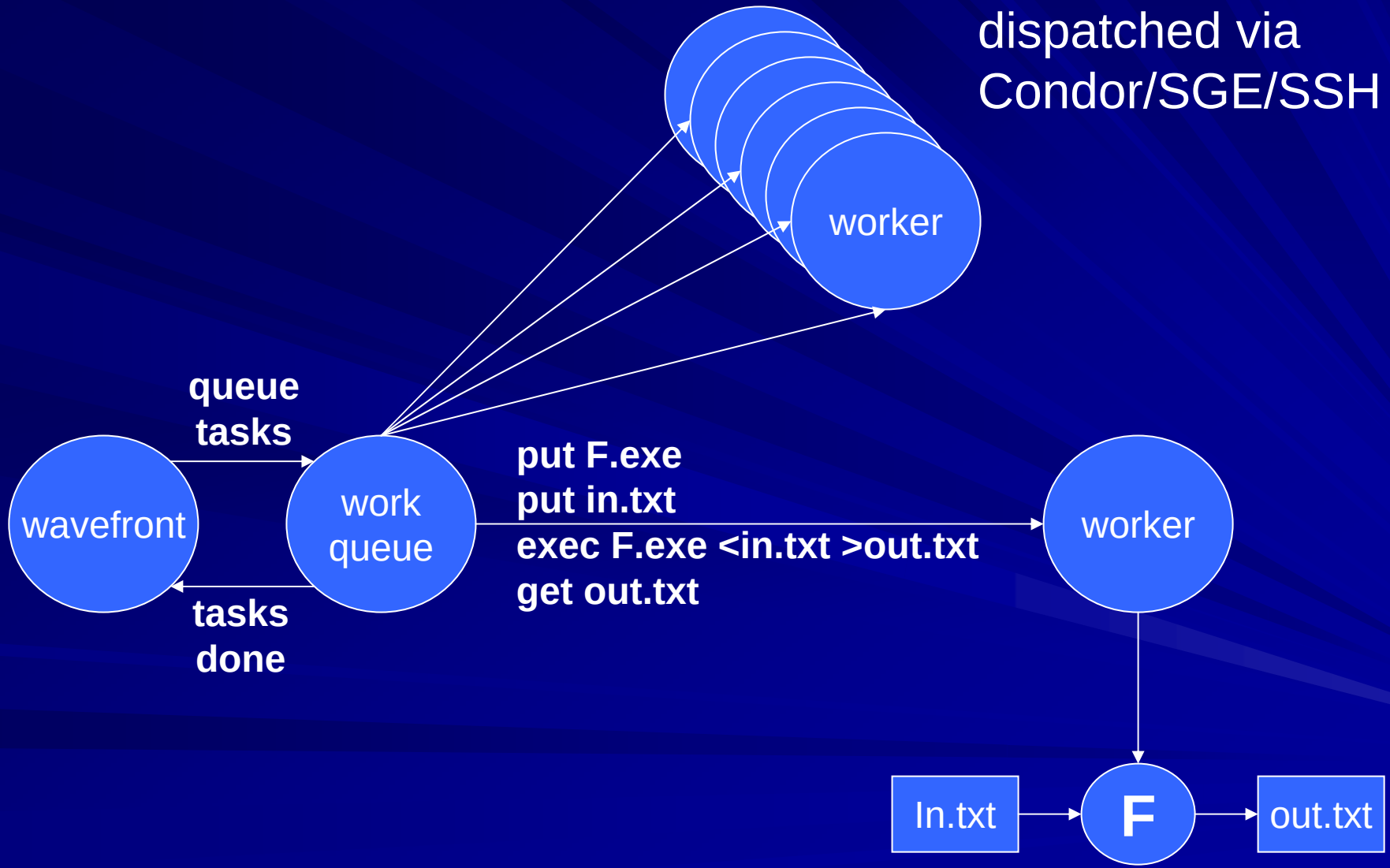
The Performance Problem

- Dispatch latency really matters: a delay in one holds up all of its children.
- If we dispatch larger sub-problems:
  - Concurrency on each node increases.
  - Distributed concurrency decreases.
- If we dispatch smaller sub-problems:
  - Concurrency on each node decreases.
  - Spend more time waiting for jobs to be dispatched.
- So, model the system to choose the block size.
- ***And, build a fast-dispatch execution system.***

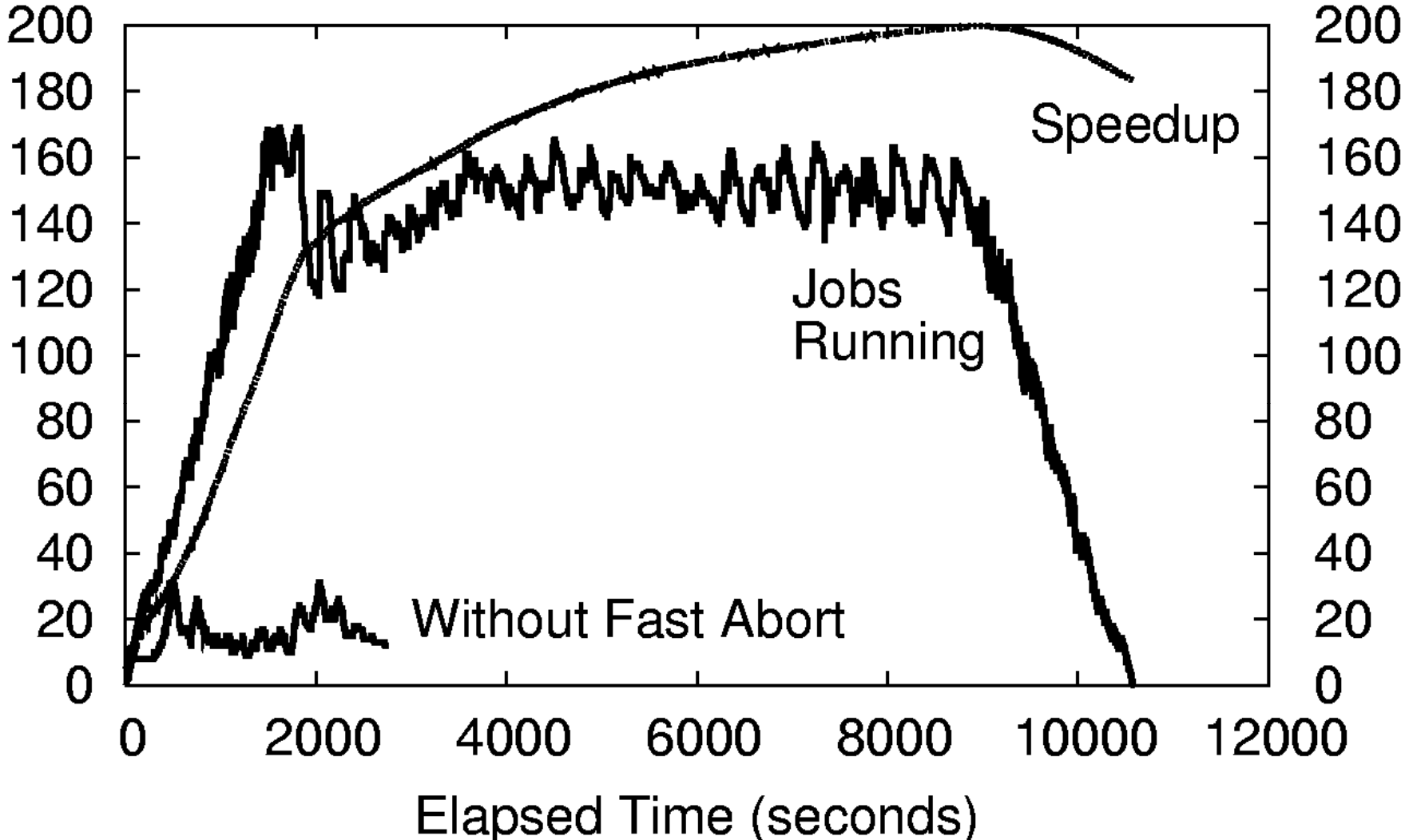
Model of 1000x1000 Wavefront



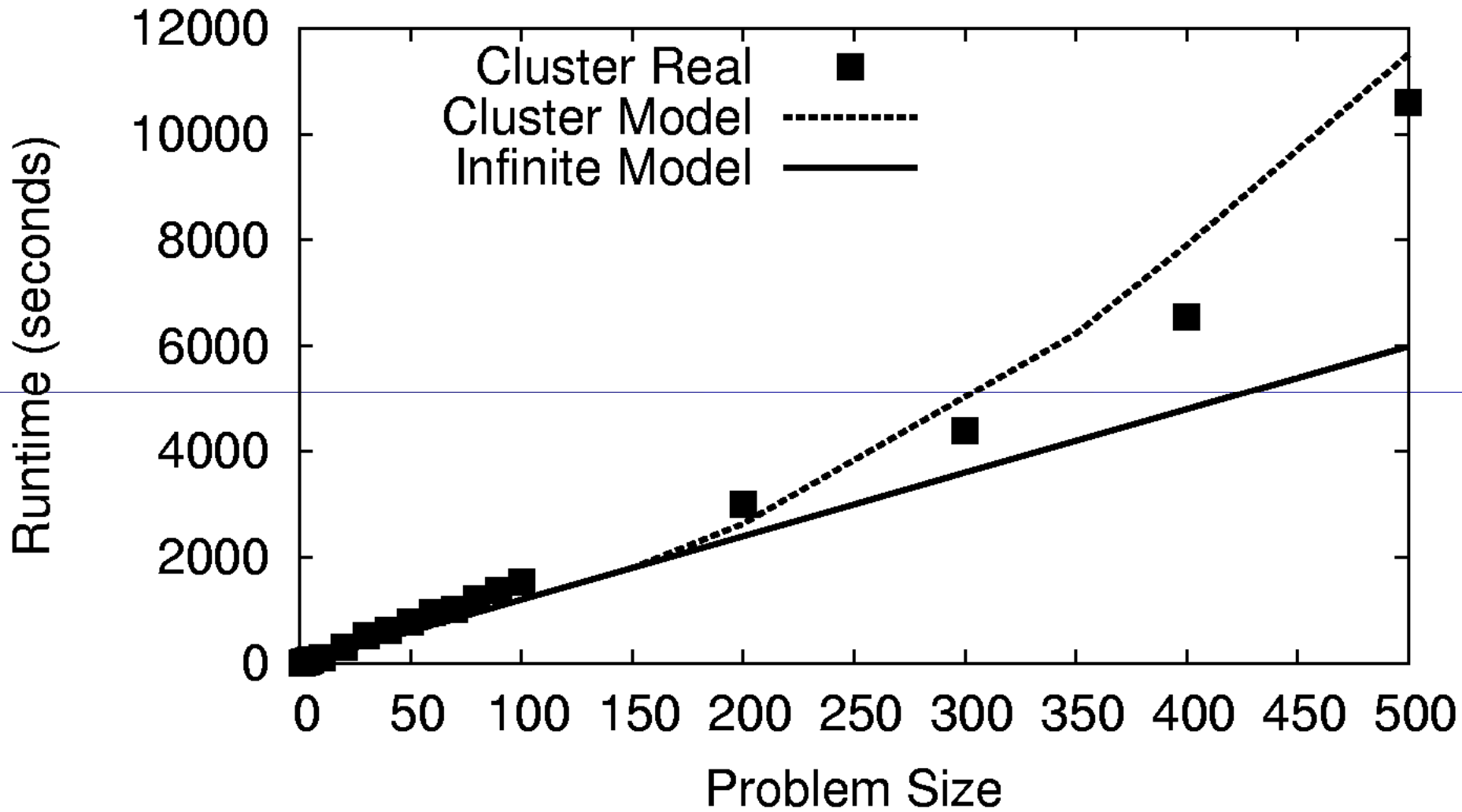
100s of workers dispatched via Condor/SGE/SSH



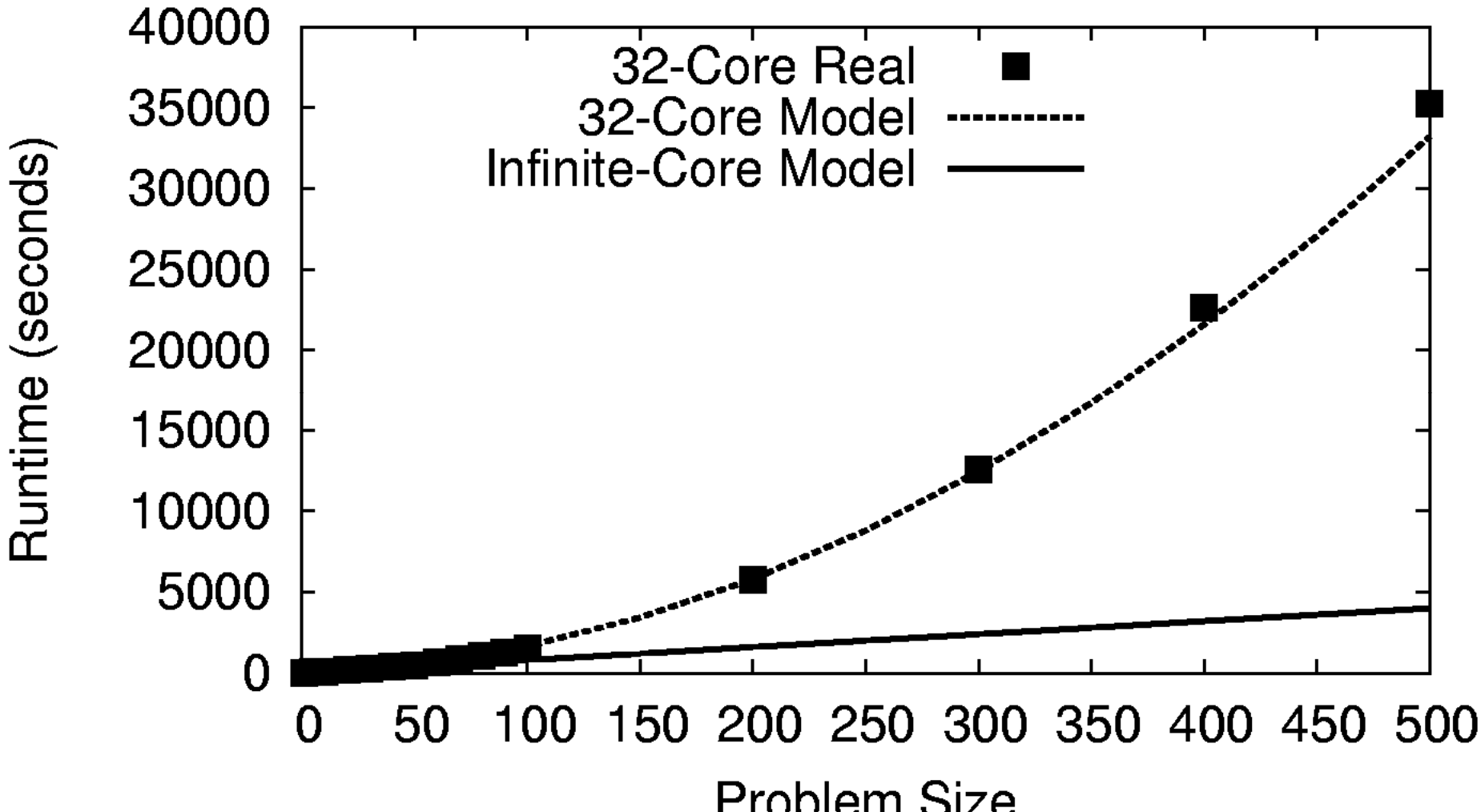
500x500 Wavefront on ~200 CPUs



Wavefront on a 200-CPU Cluster



Wavefront on a 32-Core CPU

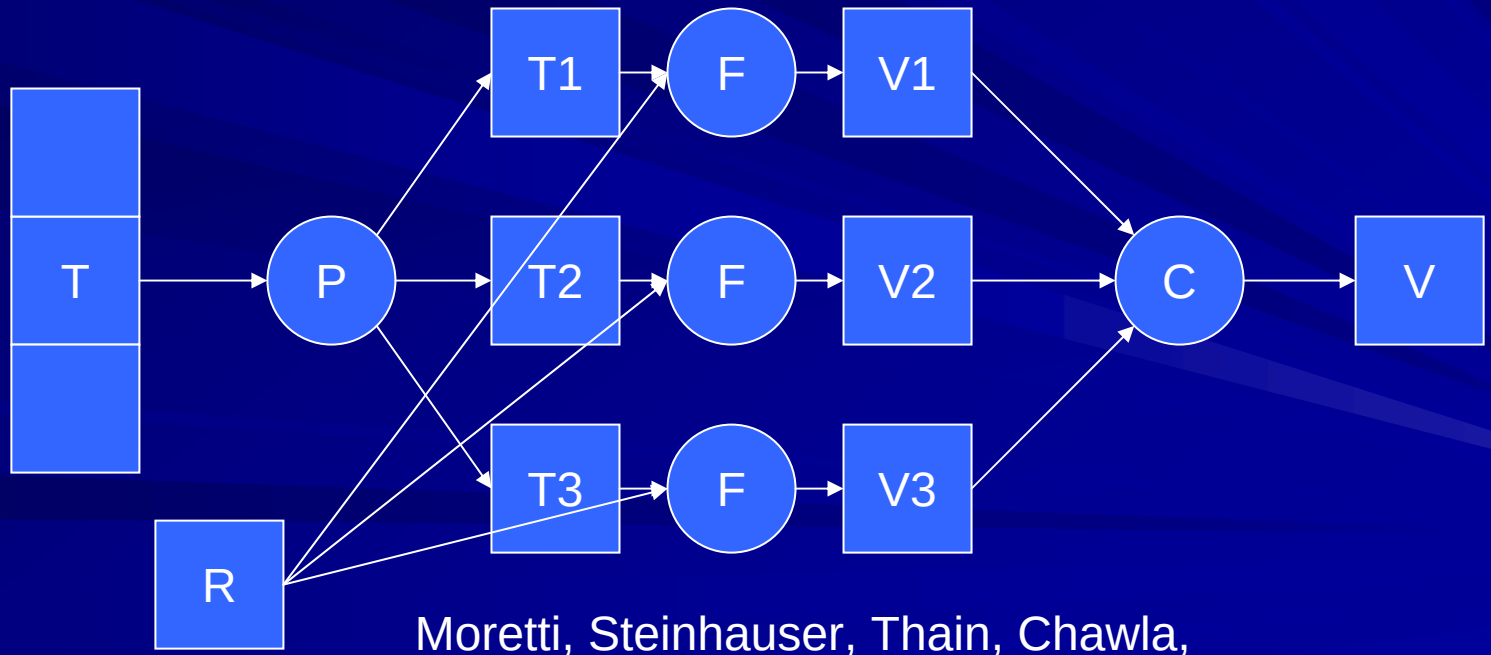


Classify Abstraction

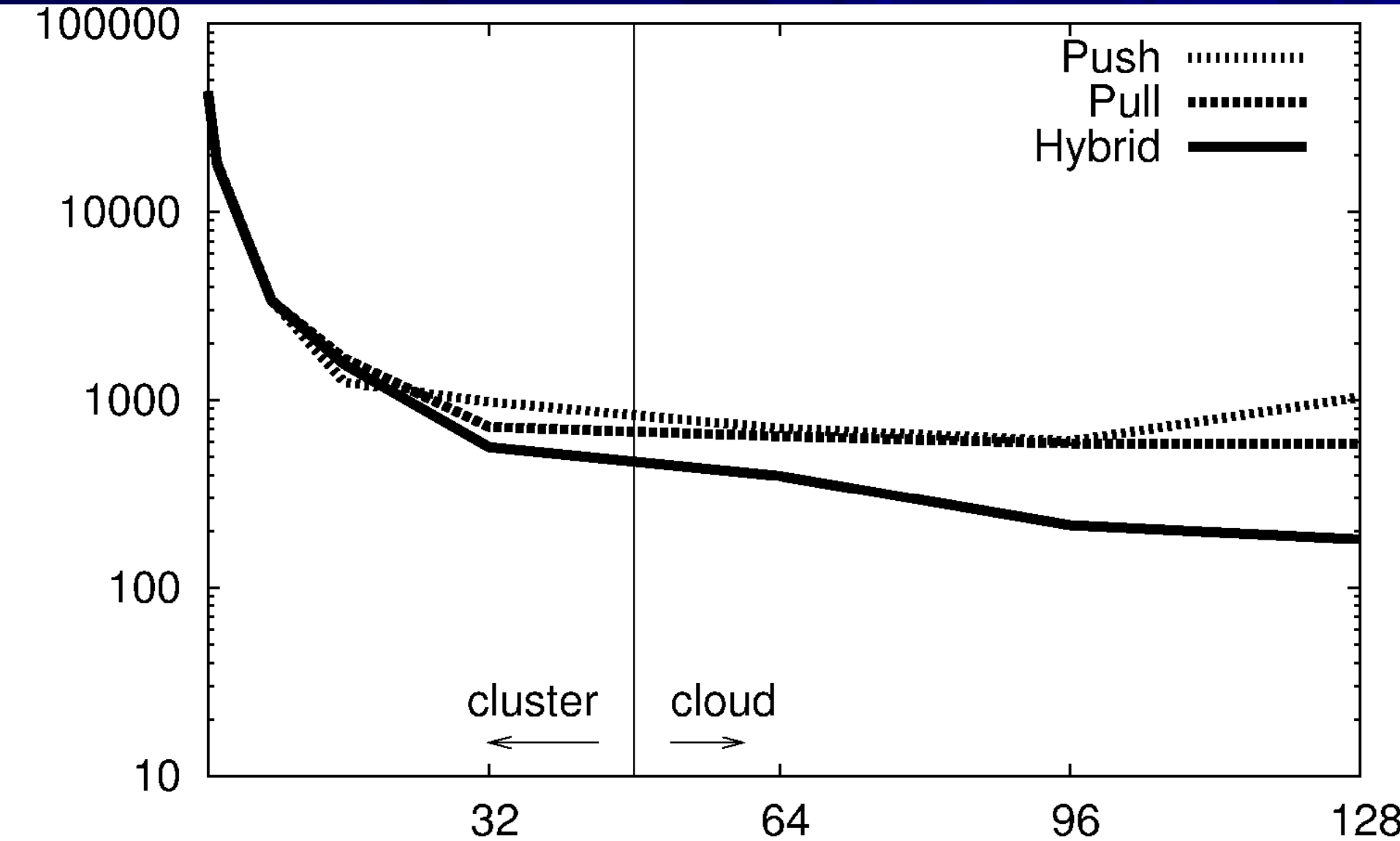
Classify(T, R, N, P, F)

T = testing set R = training set

N = # of partitions F = classifier



Moretti, Steinhauser, Thain, Chawla,
Scaling up Classifiers to Cloud Computers, ICDM 2008.


100000
10000
1000
100
10
32
64
96
128
Push
Pull
Hybrid
cluster
cloud

From Abstractions to a Distributed Language

What Other Abstractions Might Be Useful?

- Map(set S, F(s))
- Explore(F(x), x: [a….b])
- Minimize(F(x), delta)
- Minimax(state s, A(s), B(s))
- Search(state s, F(s), IsTerminal(s))
- Query(properties) -> set of objects

- FluidFlow(V[x,y,z], F(v), delta)

How do we connect multiple abstractions together?

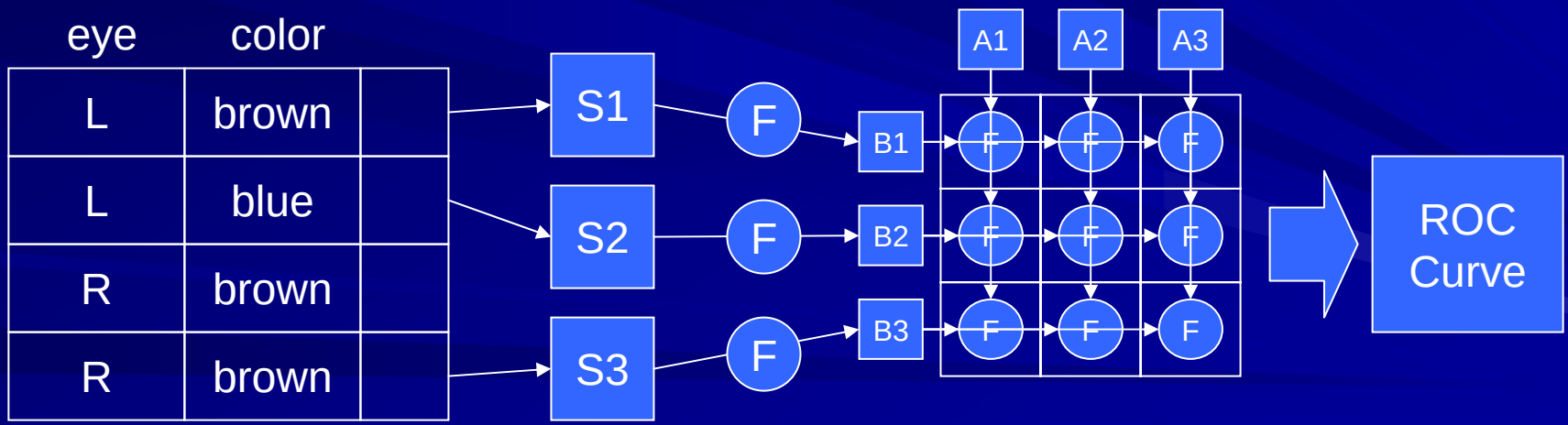
- Need a meta-language, perhaps with its own atomic operations for simple tasks:
- Need to manage (possibly large) intermediate storage between operations.
- Need to handle data type conversions between almost-compatible components.
- Need type reporting and error checking to avoid expensive errors.
- If abstractions are feasible to model, then it may be feasible to model entire programs.

Connecting Abstractions in BXGrid

S = Select(color="brown")

B = Transform(S,F)

M = AllPairs(A, B, F)



Bui, Thomas, Kelly, Lyon, Flynn, Thain
BXGrid: A Repository and Experimental Abstraction… poster at IEEE eScience 2008

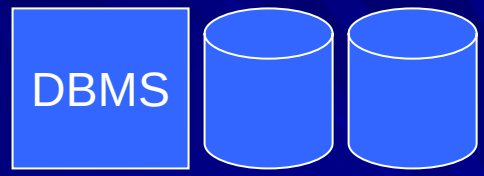
Implementing Abstractions

S = Select(color="brown")

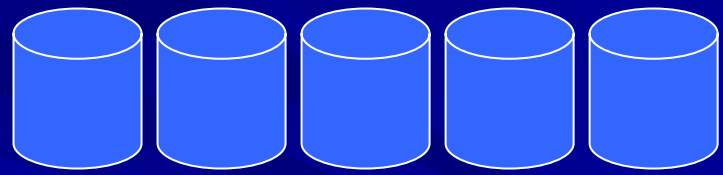
B = Transform(S,F)

M = AllPairs(A, B, F)

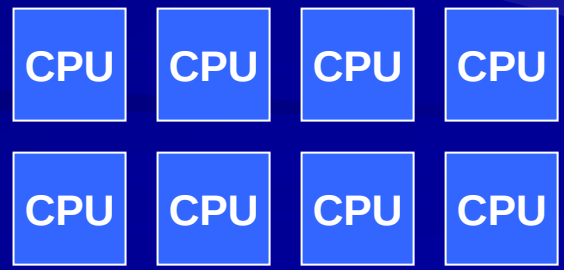
Relational Database (2x)



Active Storage Cluster (16x)



Condor Pool (500x)



BXGRID - Biometrics Research Grid - Windows Internet Explorer

https://bxgrid.cse.nd.edu/doall.php?tbl=irises_still&state=validated&where=temp_collectionid+%3D+%2712222665?

putting sql into

BXGRID - Biometrics Research Grid

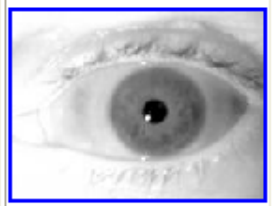
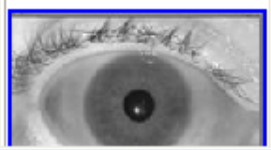
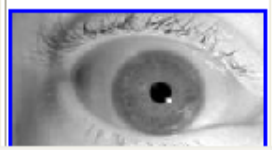
BXGRID - Biometrics Research Grid

BXGrid Main
Explore Data
+ Browse Data
+ Query Data
+ Validate Data
+ Enroll Data
Run Experiments
+ Transform Data
+ Compare Data
+ View Results
+ Manage Data
About BXGrid
+ Report Bug
+ Site Info
+ System Status
System Admin
+ Manage Bugs
+ Manage Users
dthain's Account
+ My Account
+ Logout

Data Source	State (help)	Constraint (help)	Limit	Detail	Images	Mode
iris images	validated	temp_collectionid = '122226657	10	limited	small	validate

Showing 1 to 10 of 4338 results. Download all results as TXT or CSV or XML or TGZ

Prev 10 | Next 10 | First Page | Last Page

Unvalidated	Metadata	Action	Valid 1	Valid 2	Valid 3	
 View Full Record	date 2008-09-09 00:00:00 eye Right color Brown state validated subjectid nd1S02463	Validate Problem Unvalidate	 View Full Record	 View Full Record	 View Full Record	 View Ful
 View Full Record	date 2008-09-09 00:00:00 eye Left color Brown state validated subjectid nd1S02463	Validate Problem Unvalidate	 View Full Record	 View Full Record	 View Full Record	 View Ful
	date 2008-09-09 00:00:00 eye Right color Brown	Validate Problem				

start | Welcome to th... | Gmail - Inbox (... | wizard.cse.nd.... | Microsoft Powe... | BXGRID - Biom... | 5:37 PM

What is the Most Useful ABI?

- Functions can come in many forms:
  - Unix Program
  - C function in source form
  - Java function in binary form
- Datasets come in many forms:
  - Set: list of files, delimited single file, or database query.
  - Matrix: sparse elem list, or binary layout
- Our current implementations require a particular form. With a carefully stated ABI, abstractions could work with many different user communities.

What is the type system?

- Files have an obvious technical type:
  - JPG, BMP, TXT, PTS, …
- But they also have a ***logical*** type:
  - JPG: Face, Iris, Fingerprint etc.
  - (This comes out of the BXGrid repository.)
- The meta-language can easily perform automatic conversions between technical types, and between some logical types:
  - JPG/Face -> BMP/Face via ImageMagick
  - JPG/Iris -> BIN/IrisCode via ComputeIrisCode
  - JPG/Face -> JPG/Iris is not allowed.

Abstractions Redux

- Mapping general-purpose programs to arbitrary distributed/multicore systems is algorithmically complex and full of pitfalls.
- But, mapping a single abstraction is a tractable problem that can be optimized, modeled, and re-used.
- Can we combine multiple abstractions together to achieve both expressive power and tractable performance?

Troubleshooting Large Workloads

It’s Ugly in the Real World

- Machine related failures:
  - Power outages, network outages, faulty memory, corrupted file system, bad config files, expired certs, packet filters...
- Job related failures:
  - Crash on some args, bad executable, missing input files, mistake in args, missing components, failure to understand dependencies...
- Incompatibilities between jobs and machines:
  - Missing libraries, not enough disk/cpu/mem, wrong software installed, wrong version installed, wrong memory layout...
- Load related failures:
  - Slow actions induce timeouts; kernel tables: files, sockets, procs; router tables: addresses, routes, connections; competition with other users...
- Non-deterministic failures:
  - Multi-thread/CPU synchronization, event interleaving across systems, random number generators, interactive effects, cosmic rays...

A “Grand Challenge” Problem:

- A user submits one million jobs to the grid.
- Half of them fail.
- Now what?
  - Examine the output of every failed job?
  - Login to every site to examine the logs?
  - Resubmit and hope for the best?
- We need some way of getting the big picture.
- Need to identify problems not seen before.

Job ClassAd

MyType = "Job"
TargetType = "Machine"
ClusterId = 11839
QDate = 1150231068
CompletionDate = 0
Owner = "dcieslak"
JobUniverse = 5
Cmd = "ripper-cost-can-9
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
ExitStatus = 0
ImageSize = 40000
DiskUsage = 110000
NumCkpts = 0
NumRestarts = 0
NumSystemHolds = 0
CommittedTime = 0
ExitBySignal = FALSE
PoolName = "ccl00.cse.nd
CondorVersion = "6.7.19 May 10 2006"
CondorPlatform = I386-LINUX_RH9
RootDir = "/"

User Job Log

Job 1 submitted.
Job 2 submitted.

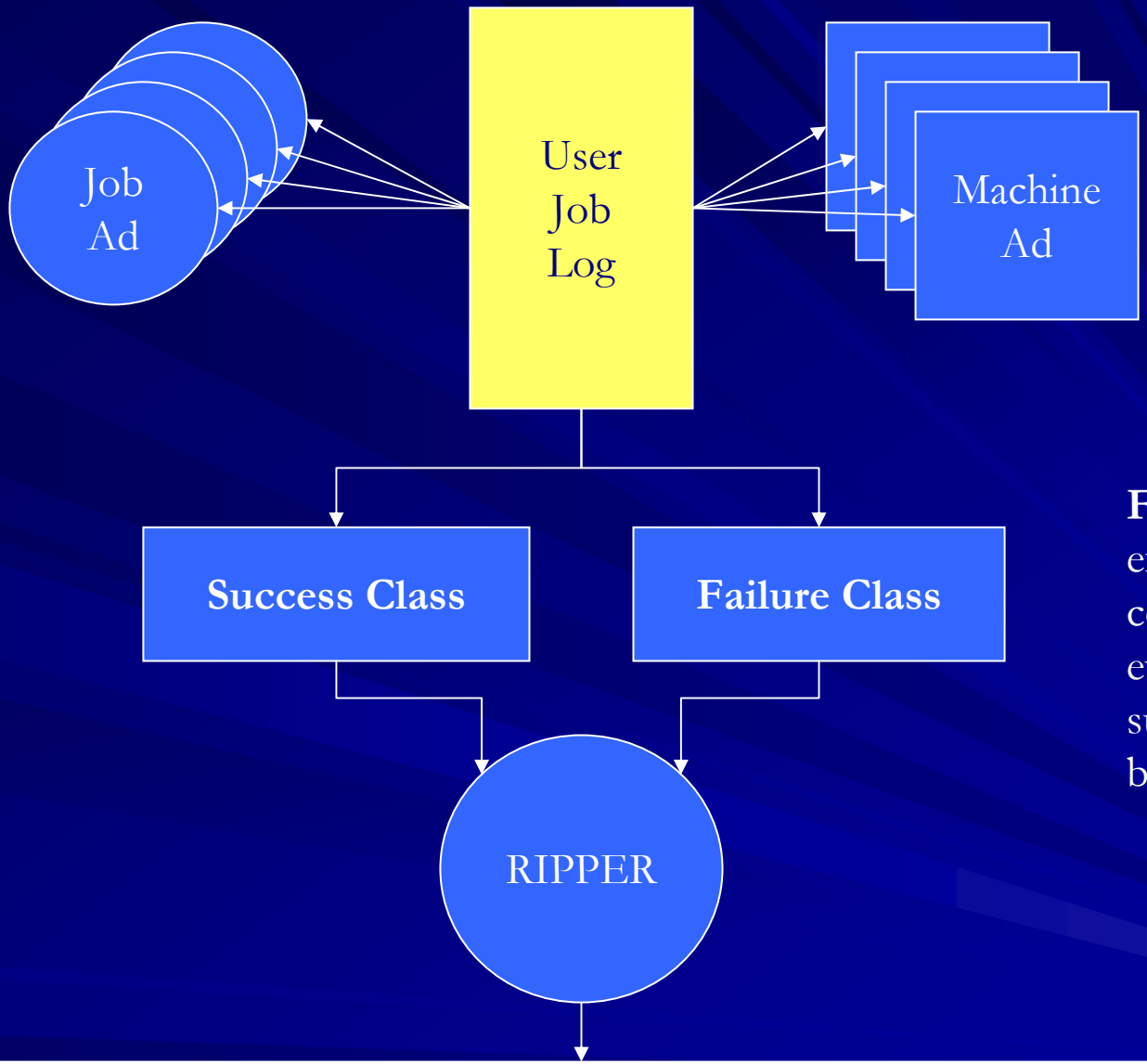
Job 1 placed on ccl00.cse.nd.edu
Job 1 evicted.
Job 1 placed on smarty.cse.nd.edu.
Job 1 completed.

Job 2 placed on dvorak.helios.nd.edu
Job 2 suspended
Job 2 resumed
Job 2 exited normally with status 1.

•••

Machine ClassAd

MyType = "Machine"
TargetType = "Job"
d.edu"
g - CondorLoadAvg)
l"
hain"
7.19 May 10 2006"
386-LINUX_RH9"
1
000
2948
.000000
KeyboardIdle = 817093
ConsoleIdle = 817093
StartdIpAddr = "<129.74.153.164:9453>"



Failure Criteria:
exit !=0
core dump
evicted
suspended
bad output

Your jobs work fine on RH Linux 12.1 and 12.3 but they always seem to crash on version 12.2.

---------- run 1 ----------
Hypothesis:
exit1 :- Memory>=1930, JobStart>=1.14626e+09, MonitorSelfTime>=1.14626e+09 (491/377)
exit1 :- Memory>=1930, Disk<=555320 (1670/1639).
default exit0 (11904/4503).
Error rate on holdout data is 30.9852%
Running average of error rate is 30.9852%
---------- run 2 ----------
Hypothesis: exit1 :- Memory>=1930, Disk<=541186 (2076/1812).
default exit0 (12090/4606).
Error rate on holdout data is 31.8791%
Running average of error rate is 31.4322%
---------- run 3 ----------
Hypothesis:
exit1 :- Memory>=1930, MonitorSelfImageSize>=8.844e+09 (1270/1050).
exit1 :- Memory>=1930, KeyboardIdle>=815995 (793/763).
exit1 :- Memory>=1927, EnteredCurrentState<=1.14625e+09, VirtualMemory>=2.09646e+06, LoadAvg>=30000, LastBenchmark<=1.14623e+09, MonitorSelfImageSize<=7.836e+09 (94/84).
exit1 :- Memory>=1927, TotalLoadAvg<=1.43e+06, UpdatesTotal<=8069, LastBenchmark<=1.14619e+09, UpdatesLost<=1 (77/61).
default exit0 (11940/4452).
Error rate on holdout data is 31.8111%
Running average of error rate is 31.5585%

Unexpected Discoveries

- Purdue (91343 jobs on 2523 CPUs)
  - Jobs fail on machines with (Memory>1920MB)
  - Diagnosis: Linux machines with > 3GB have a different memory layout that breaks some programs that do inappropriate pointer arithmetic.
- UND & UW (4005 jobs on 1460 CPUs)
  - Jobs fail on machines with less than 4MB disk.
  - Diagnosis: Condor failed in an unusual way when the job transfers input files that don't fit.

Condor Log Analyzer

Upload Your Log Files:

Required User Log File: Browse...

Optional Machine File: Browse...

Upload My Log Files

Frequently Asked Questions

- **What's going on here?**
  This web site allows you to upload log files generated by the Condor distributed computing system, and get back graphics and an explanation of what happened in the system. This can aid in understanding a workload of hundreds or thousands of jobs. Here is an example of the output.

- **How do I create a user log file?**
  Add this line to your Condor submit file: `log = userlog.txt`

- **How do I create a machine file?**
  Run this command: `condor_status -l > machinefile.txt`

- **Why are you doing this?**
  We are constructing new tools that help people to debug distributed systems. The problem is, we need lots of log data to test our ideas on. So, if you upload your log files, you get (we hope) a useful result, and we get more data to practice on.

- **Will others be able to see my data?**
  The URL of your results is based on the checksum of your logfile. So, if you **want** to share your results with others, you can simply send the URL to your friends. If you **do not** want others to see your data, then don't publicize the URL, and it is highly unlikely anyone could guess it.

- **Can I download graphics and logs to use in my own papers, presentations, etc?**
  Yes, go right ahead. We would appreciate it if you give us a credit with the following citation: *Douglas Thain, David Cieslak, and Nitesh Chawla, "Condor Log Analyzer", http://condorlog.cse.nd.edu, 2009.*

Condor Log Analyzer

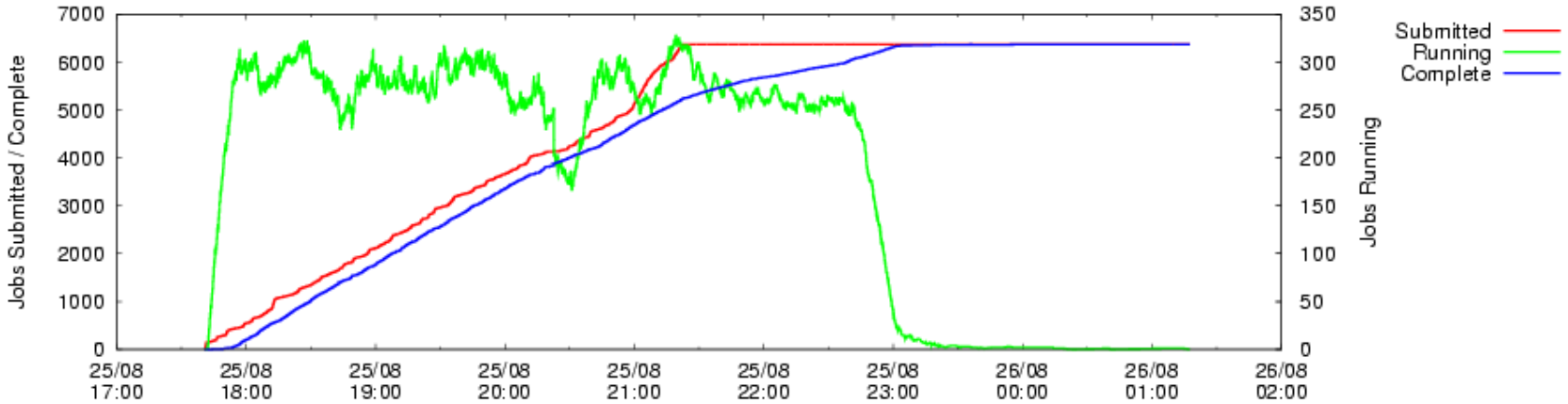
Workload Summary:

CPU Time: 60+18:22:56
Total Goodput: 57+09:39:27
Total Badput: 3+08:43:29
Elapsed Time: 0+07:36:44

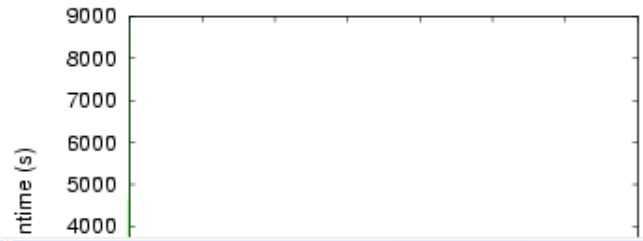
Event Summary:

Jobs Submitted: 6372
Execute Events: 7812
Vacate Events: 1855
Jobs Removed:
Jobs Complete: 6372

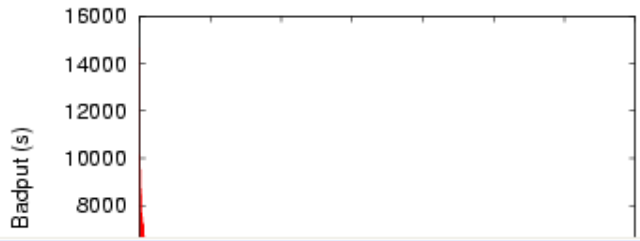
Workload Timeline - gif - eps - gnuplot - data



Runtime by Job - gif - eps - gnuplot - data



Badput by Job - gif - eps - gnuplot - data



Five Fastest Jobs: (see all jobs)

```
37 (324042.000.000) "slot2@sc0-29
37 (325716.000.000) "slot2@sc0-26
38 (324387.000.000) "slot2@sc0-11
38 (325030.000.000) "slot2@sc0-16
38 (326076.000.000) "slot2@sc0-03
```

Slowest Jobs: (see all jobs)

Acknowledgments

- Cooperative Computing Lab
  - http://www.cse.nd.edu/~ccl
- Faculty:
  - Patrick Flynn
  - Nitesh Chawla
  - Kenneth Judd
  - Scott Emrich
- Grad Students
  - Chris Moretti
  - Hoang Bui
  - Karsten Steinhauser
  - Li Yu
  - Michael Albrecht
- Undergrads
  - Mike Kelly
  - Rory Carmichael
  - Mark Pasquier
  - Christopher Lyon
  - Jared Bulosan
- NSF Grants CCF-0621434, CNS-0643229