# Desktop Grids

ICS 691

# Desktop Grids

- Although clusters are relatively cheap and mainstream, an even cheaper and easier alternative would be great
- How about reusing desktop resources that
  - are already purchased
  - are distributed so don't require infrastructure
    - space, power, A/C
- We can put them all together in Desktop Grids
- Question: where do we find these resources?
- Answers:
  - In people's home: "Internet Desktop Grids"
    - Question: what is the incentive?
  - In corporations: "Enterprise Desktop Grids"
    - May use desktop machines AND clusters

# Internet vs. Enterprise

- Most well-known projects are for Internet-wide computing
  - Humanitarian/Fun/Geeky applications
- Some start-up companies tried to sell a compute service using machines at people's home
  - Many failed
  - Reason: people don't want to have their idle cycles used just for anything
  - Even if one pays their cable bill!
- These companies had to adapt to the Enterprise environment
  - Convince a CEO that buying the software will make it possible to get better return on investment for the thousands of desktop machines purchased
- Main company today: United Devices
  - Spin-off of SETI@home
- Academic projects:
  - Condor
  - BOINC
  - XtremWeb

# Desktop Grids: The Largest Distributed Computing Systems

SETI@HOME 67 TFlops/sec, 500,000 workers, $700,000

GIMPS 17.5 TFlops/sec, 80,000 workers

Folding@home distributed computing 186 TFlops/sec, 195,000 workers

climateprediction.net

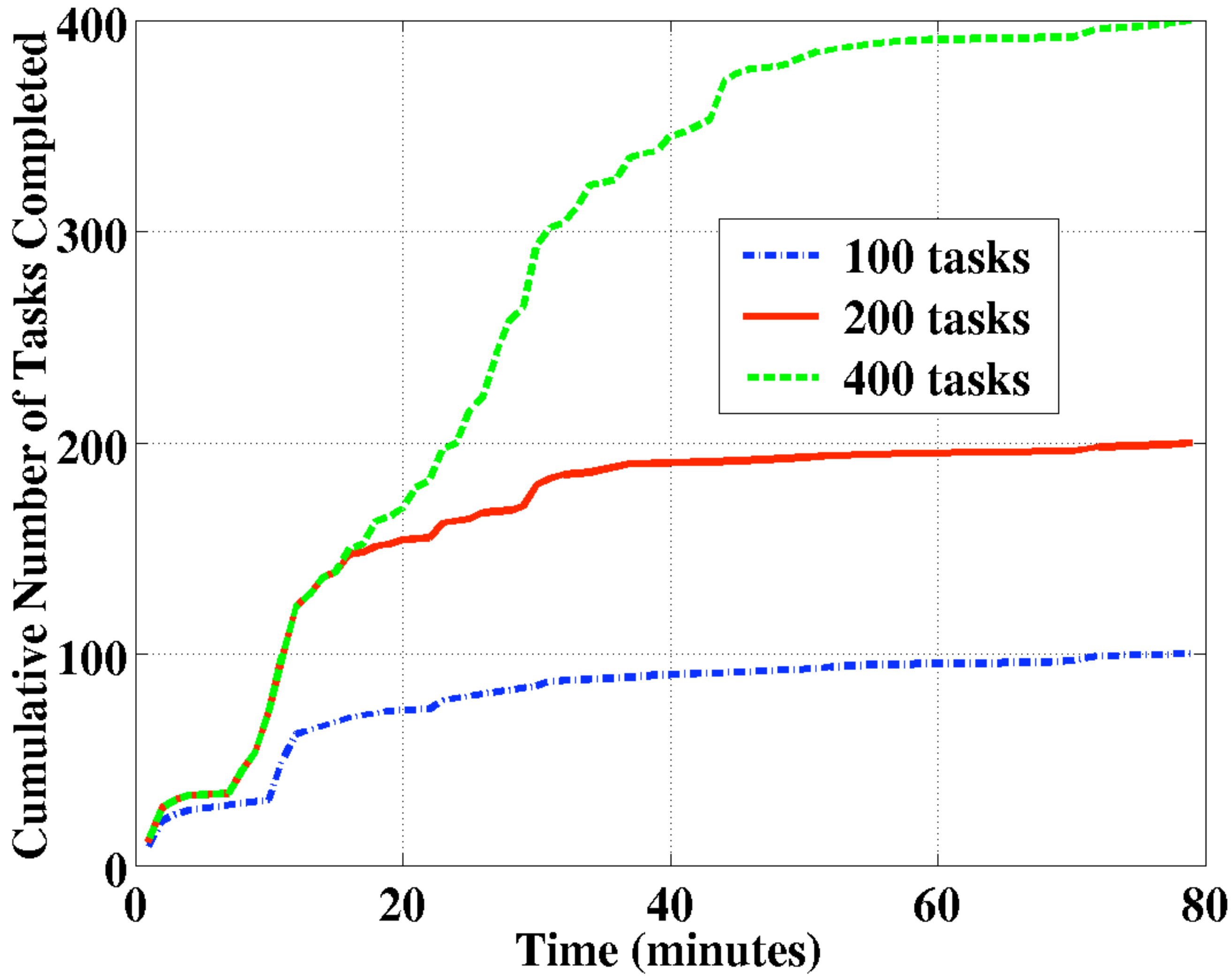
EINSTEIN@HOME Catch a Wave From Space

distributed.net

# High-throughput Computing

- Desktop Grids are typically used for “high-throughput”, compute-intensive applications
- High-Throughput?
  - Many individual, independent tasks
  - The performance metric is the task completion rate over “long” periods of time (e.g., month)
    - As opposed to makespan
- Implication: The “waiting for the last task” problem goes away
  - Simple scheduling heuristics such as FCFS can be effective when there are many tasks

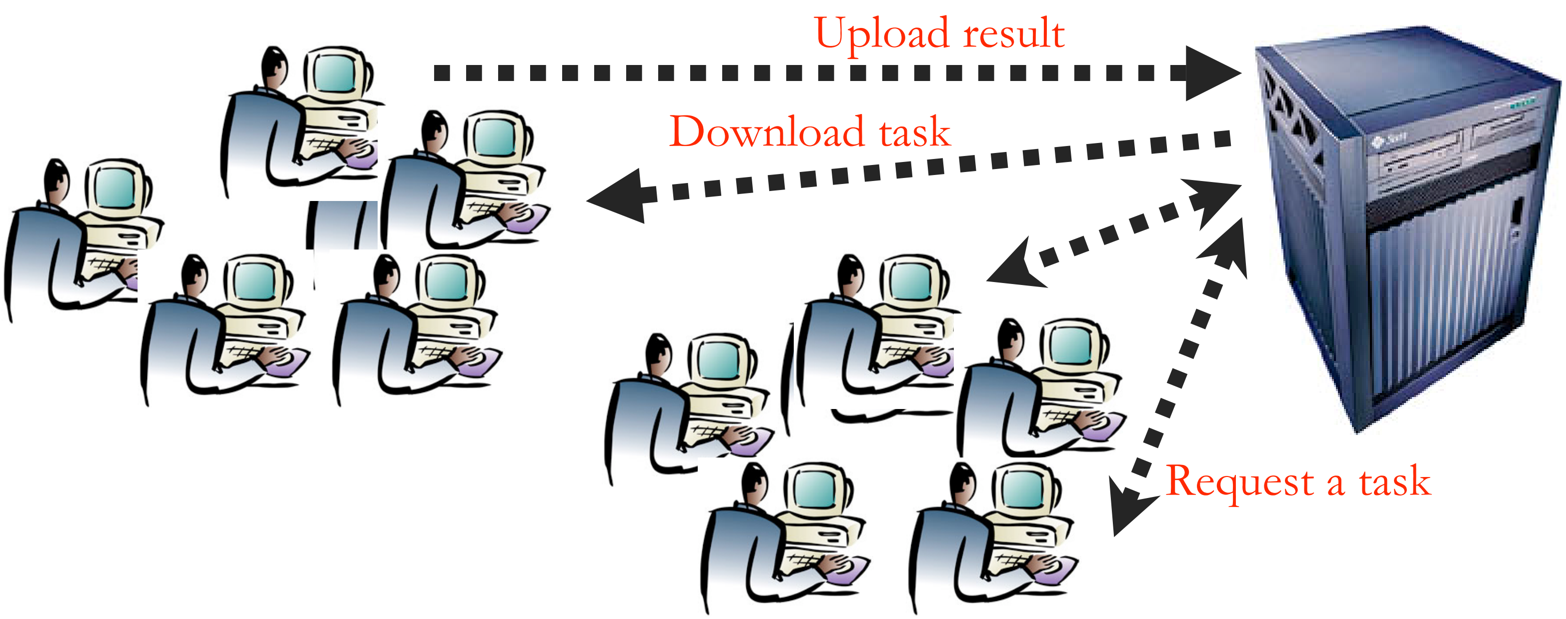
# FCFS Scheduling




200 hosts

# Desktop Grid Background

- Simple model

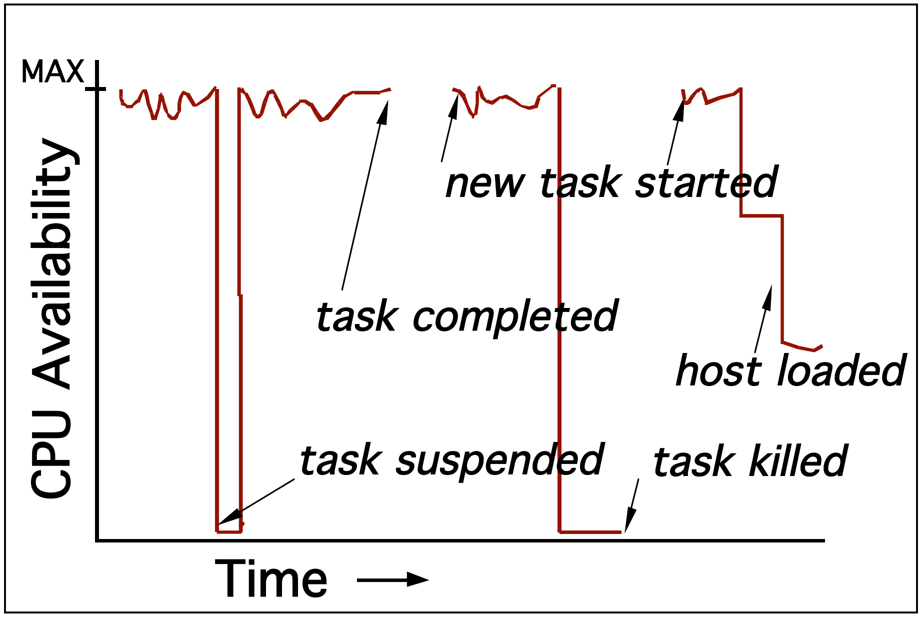


# Desktop Grid Client

- Implementation
  - Embedded in a screen saver
  - As a stand-alone daemon
- The resource owner can typically
  - Disable the client
  - Set resource consumption limits for the client
    - Run only when I am not using more than 10% of the CPU
    - Run only when I am not running any program
    - Run only between midnight and 6AM
    - etc.
- Note the client/server terminology
  - the client performs computation for the server!

# Desktop Grid Resources

- Resources shared: unreserved, volatile
  - Variable CPU load
  - Variable host availability



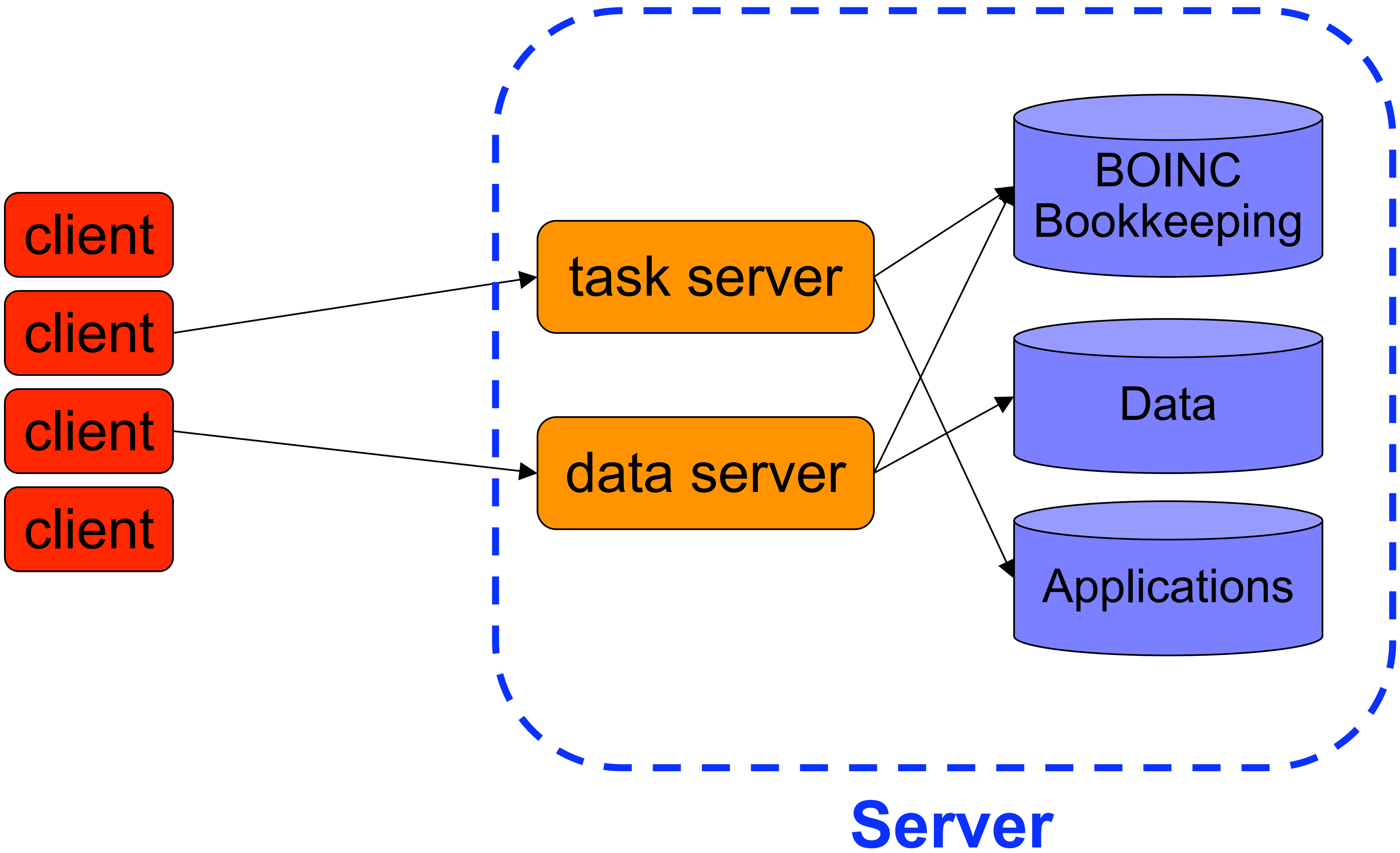
# SETI@Home & BOINC

- The most famous Internet Desktop Grid application is SETI@Home
  - Processes data from the Arecibo Radar Telescope array
  - Attempts to detect “Alien” patterns in the data
  - Gathered more than half-million clients
  - In fact too many resources for its needs
  - Many clients just perform redundant work!
- Has provided the blueprint for how to do Internet desktop grids
  - Was the basis for the “United Device” company
  - Was the basis for the BOINC Project

# BOINC

- BOINC: Berkeley Open Infrastructure for Network Computing
- Supports many applications
  - At the server
  - At the client
- Participants have “low” connectivity
  - Applications have large computations
  - Applications have small data
- Participants’ machines fail or just never come back
- Client may be hacked and be malicious
  - Denial of service
  - Forged results (“I found ET!”)

# BOINC: Centralized



# BOINC Security

- Result falsification
  - Task replication to achieve consensus
- DoS attacks
  - limited upload sizes
  - signed results
  - failures: exponential back off
- No sandboxing at the client level
  - Applications had better be correct and non-malicious

# Sandboxing

- Several options to guarantees that a client machine is safe
- Disallow system calls
  - Provide own API for “sytem call” type things
  - Burden to the application writer
- Build and use a Virtual Machine
  - XtremWeb does this
  - A lot of work but allows best control
- Use the JVM as a virtual machine
  - But one is restricted to Java applications
- System call sandboxing
  - Intercept system calls
  - Check them or simulate them
  - High overhead

# Falsified Results

- Here again, there are several possible techniques
- Spot Checking
  - once in a while, send out a work unit whose result is known
  - blacklist clients that send back a wrong answer
    - And any past results from that client are discarded
  - Minimal redundant computation
  - But is blacklisting even possible in an Internet environment?
- Majority voting
  - There are theoretical studies on the trade-offs between redundancy and probability of detecting erroneous results
- Credibility based schemes
  - Keep track of how good a client has been in the past
  - Not waste redundant/useless computation on good clients all the time

# Enterprise Desktop Grids

- Although Internet-wide desktop grids are interesting and popular, they have many drawbacks
  - That lead to interesting/fun questions
  - But that may not be what a company wants to deal with
- In an Enterprise, many issues go away
  - Less heterogeneity?
  - Fewer security issues?
  - Better networks?
  - Machines never turned off?
  - Better machines?
  - More intensive applications
    - More data
    - More computation

# Typical “Enterprise” Desktop Grid Applications*

| Application | Task run time | Task data size | Server bandwidth |
|---|---|---|---|
| Docking | 20 min. | 1 MB | 6.67 Mbps |
| Small data, med run | 10 min. | 1 MB | 13.3 Mbps |
| BLAST | 5 min. | 10 MB | 264 Mbps |
| Large data, large run | 20 min. | 20 MB | 132 Mbps |

*Grid Resource Management, Chapter 26: Resource Management in the Entropia System

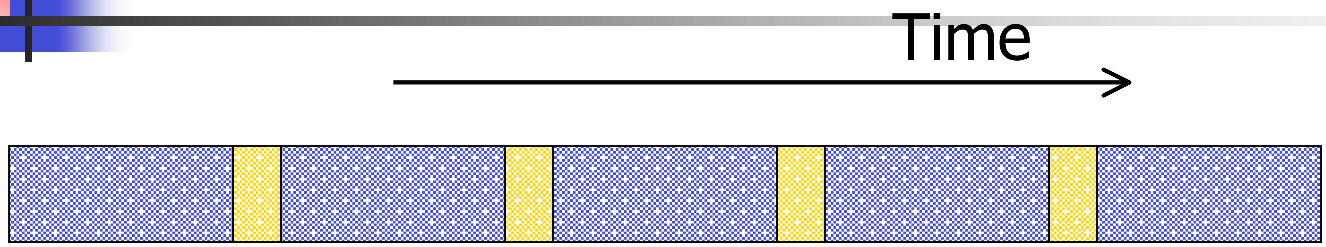
# Condor

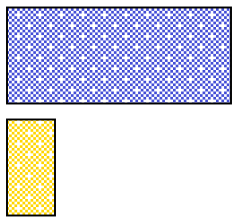
- Condor: a Hunter of Idle Workstations
- Old project still used today (started in 1985)
  - Many “Condor Pools” in many institutions
- Targets sets of machines in universities
  - Clusters
  - Student labs
  - Workstations
- Provides a job submission mechanisms like a batch scheduler
  - No concept of a server that stores specific applications
  - resource owners can still specify usage constraints
- Users can specify job dependencies
- Users can specify job resource requirements
- The matchmaker matches jobs with resources
- Condor is like its own Grid infrastructure
  - In fact, it provides a gateway to Globus

# Checkpointing

- What happens when a task gets killed?
- One option is that the task is lost and must be restarted from scratch
  - An viable option if tasks are short compared to "availability intervals"
- Another option is to do what's called "checkpointing"
  - Checkpointing: save the task's state prediodically, so that if killed, the task can be restarted from the last checkpoint
  - Condor can do this

# Checkpointing

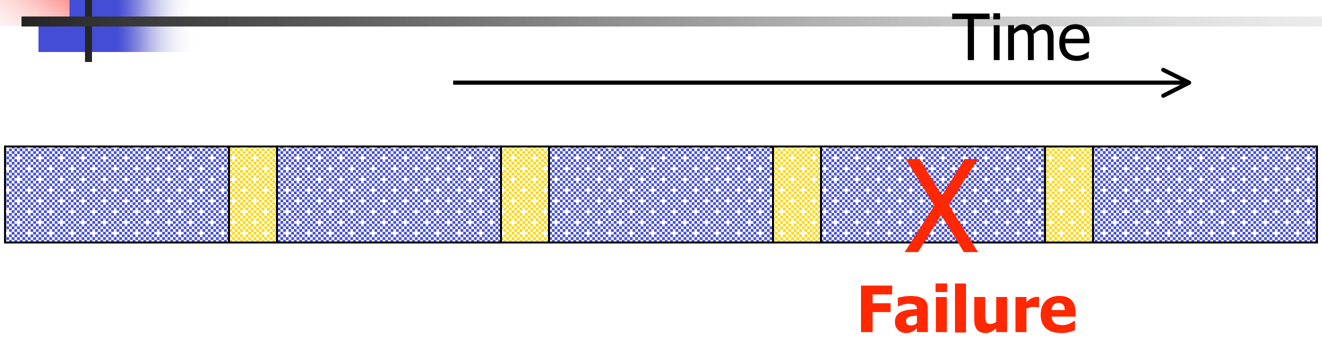


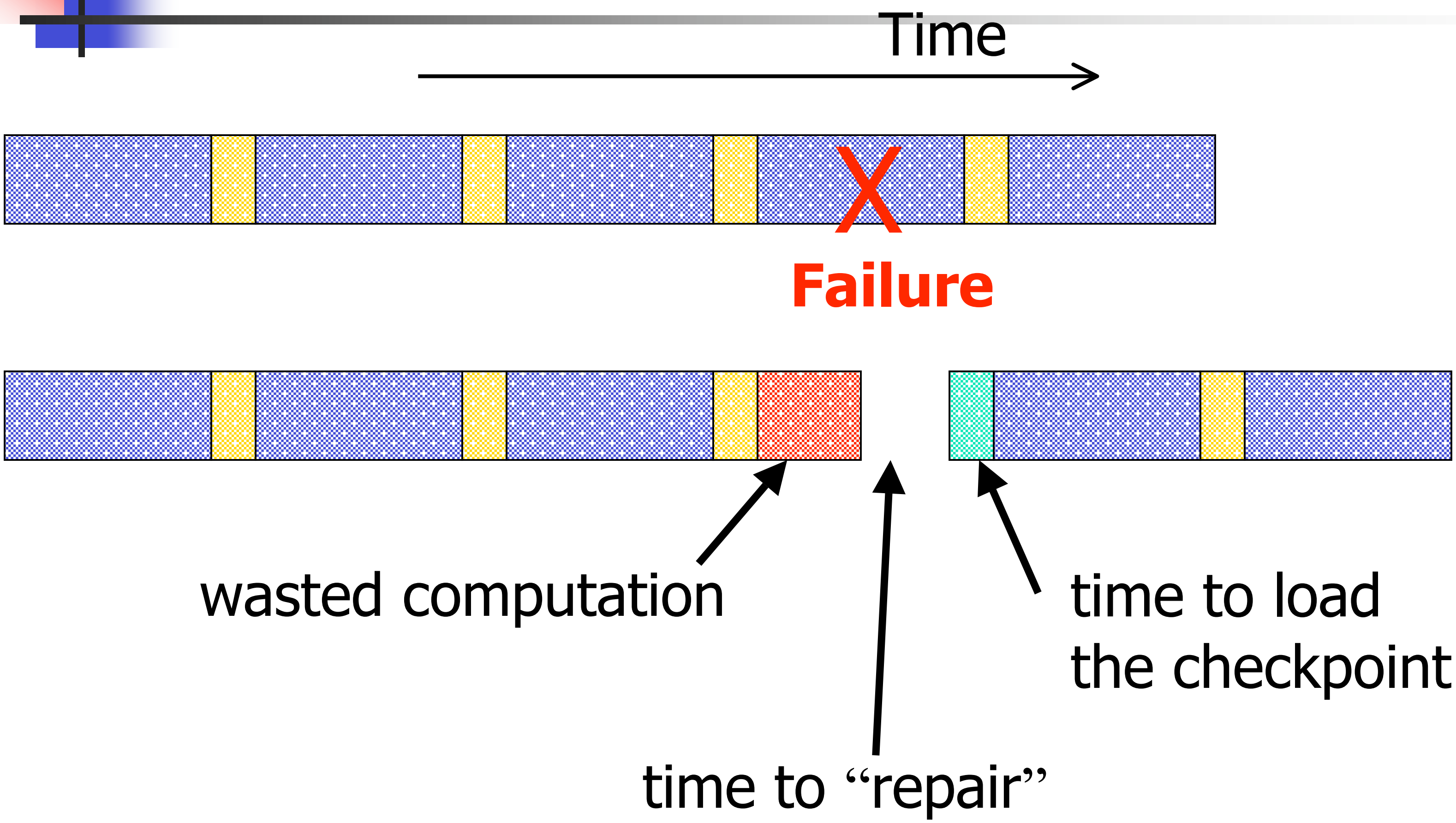
Do useful computation

Save application's state

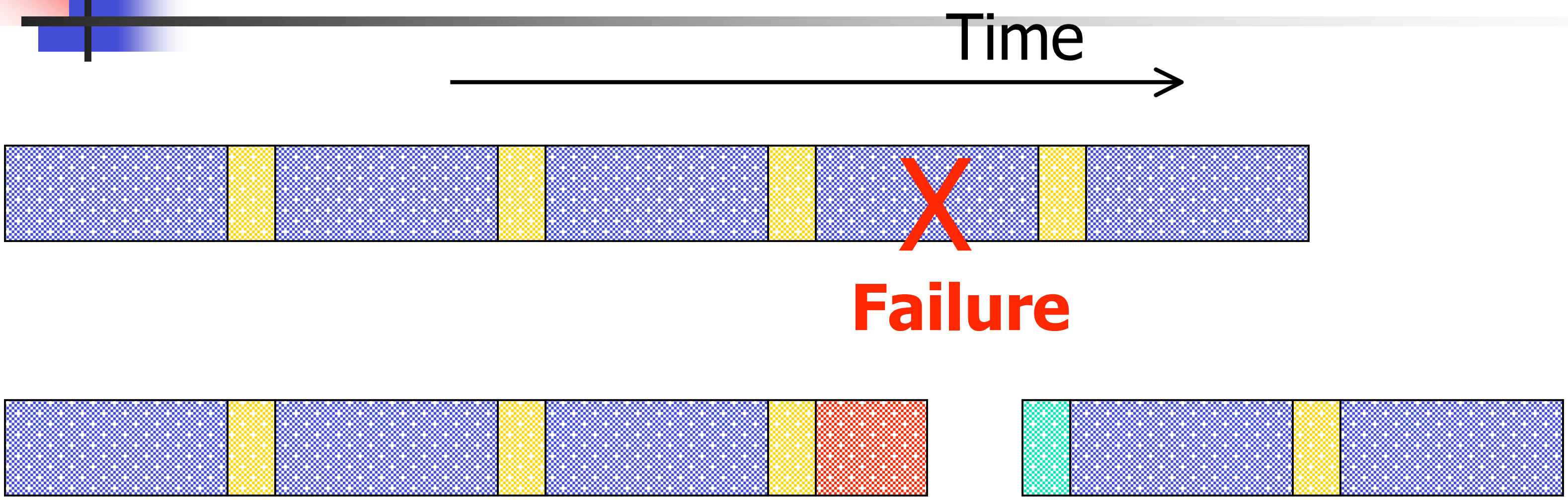
# Checkpointing



# Checkpointing



# Checkpointing




- Given the time to checkpoint, the time to load from a saved checkpoint, the expected time to repair, and the expected time to failure, one can figure out the best (statistically) checkpointing frequency

# Two kinds of checkpointing

- Application-level checkpointing
  - the application just periodically opens a file and saves important state in it
    - e.g., save the matrix at the previous iteration as well as the current iteration number
  - the application can be started in “recover from checkpoint mode”
    - e.g., load the matrix and the current iteration number from a file
- System-level checkpointing
  - just dump the whole memory of the process to a binary file
    - heap, stack, data segment, etc.
  - use the O/S to restart from the dumped state

# Checkpointing trade-offs

- Application-level Checkpointing
  - Saves only the data that must be saved
  - Is portable across architectures
    - In case one needs to migrate the application
  - Can require quite a bit of work to port an application to a desktop grid
  - Some desktop grid systems provide a checkpointing API

- System-level Checkpointing
  - Requires no application code modification
    - which could be cumbersome
  - Checkpointing can happen at any point in the code
  - Requires linking to a special “checkpoint” library
    - May preclude the use of some system calls
  - Condor provides such a library
  - Can only work in heterogeneous environments
    - Not good/useful for something like SETI@home

# Checkpointing and Desktop Grids

- Application-level checkpointing
  - Typically for grids that run only a few registered applications (BOINC)
  - Would allow migration even in a heterogeneous grid, but isn't typically done
    - local checkpointing only
    - no checkpointing server
- System-level checkpointing
  - Done by enterprise grids where resources are more or less homogeneous (Condor)
  - Allows migration as long as there is a checkpoint server
  - Only feasible for applications that can live without some system calls
- No checkpointing at all
  - The desktop grid infrastructure is not aware of any application checkpointing
    - Some may occur unbeknownst to the infrastructure
  - Simplifies the desktop grid infrastructure
  - More common than one would think
  - When a task fails, just restart it

# DG or cluster?

- Question:
  - I have a 200-node desktop grid
    - Perhaps in my corporation
    - Let us assume no checkpointing
  - I have an embarrassingly parallel application and I care about high throughput
  - Would I be better off with a 16-node cluster?
- To answer this question one must find out what a desktop grid may look like
- Based on desktop grid measurements

# Desktop Grid Measurements

- What we need to measure is: how many CPU cycles per hour are available on a typical desktop grid

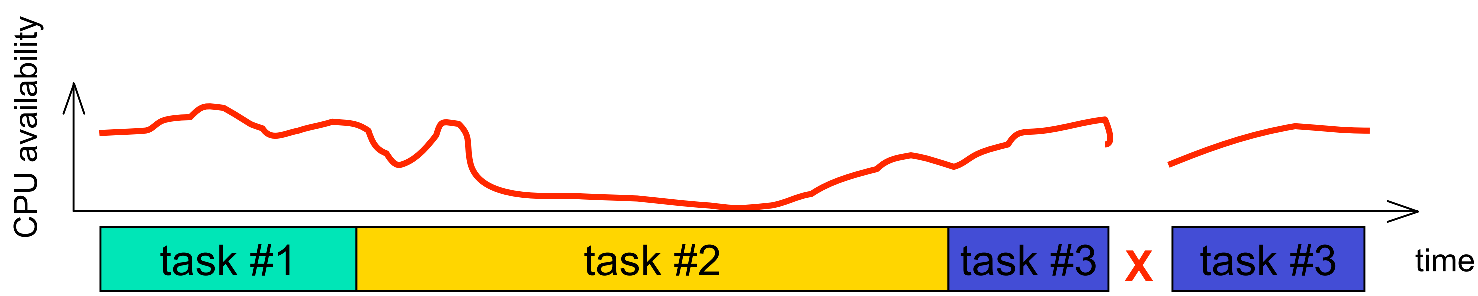
- We want to observe desktop grids and obtain trace data
  - Trace data can be used to drive simulation experiments
  - Useful for developing predictive, generative, or explanatory models, such as comparing a desktop grid with a cluster

# Previous work in the area

- Host availability [Wolski03, Long95, Bhagwan03]
  - host up / how down
  - Hard to relate uptimes to actually CPU availability
- Monitored CPU availability/load [Livny91, Wolski99, Dinda98, Arpaci95, Bolosky00]
  - Network Weather Service (NWS)
  - Difficult due to OS idiosyncrasies
- Besides
  - these methods ignore keyboard/mouse activity
  - these methods ignore the resource owner affecting the client

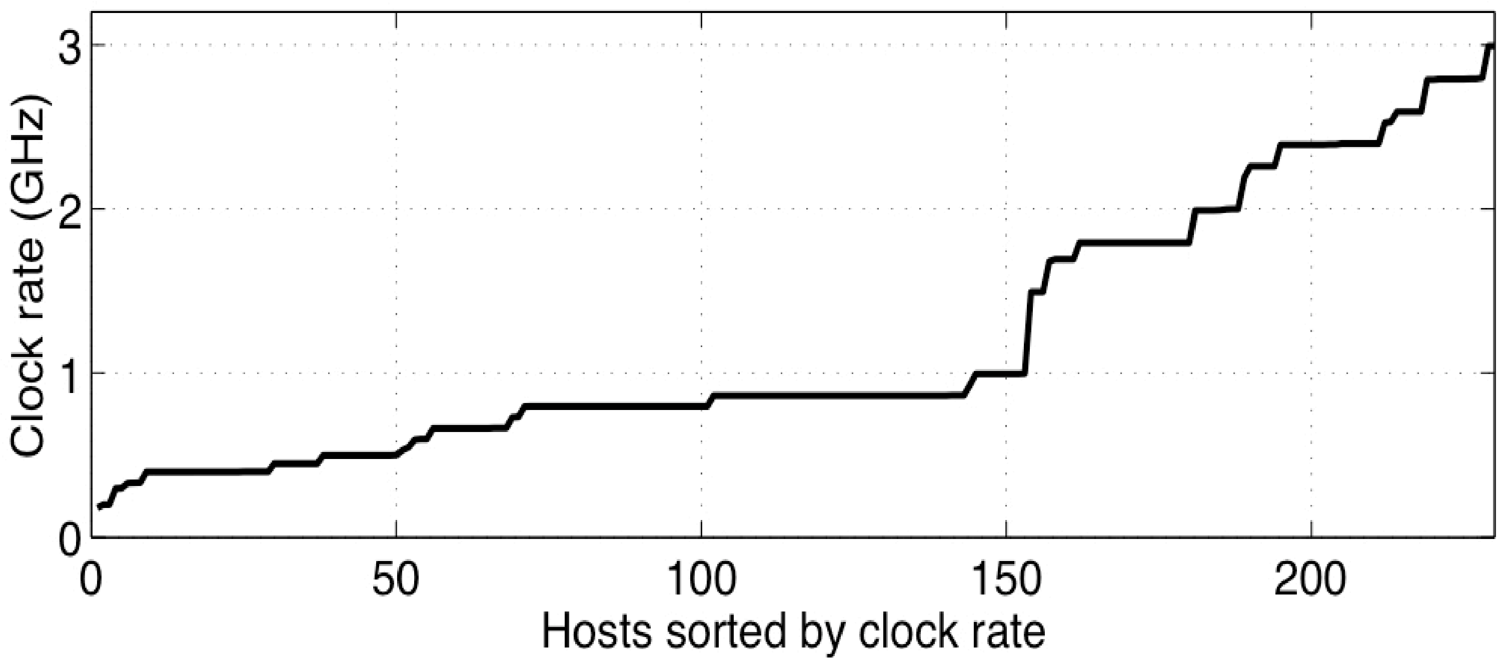
# Desktop Grid Measurements

- Observe host and CPU availability exactly as any real desktop grid application would
- Submit infinite series of tasks to a desktop grid
  - Task continuously compute a mix of floating point/integer operations and write number of completed operation every 10 secs to file
  - Tasks do not interfere with desktop user



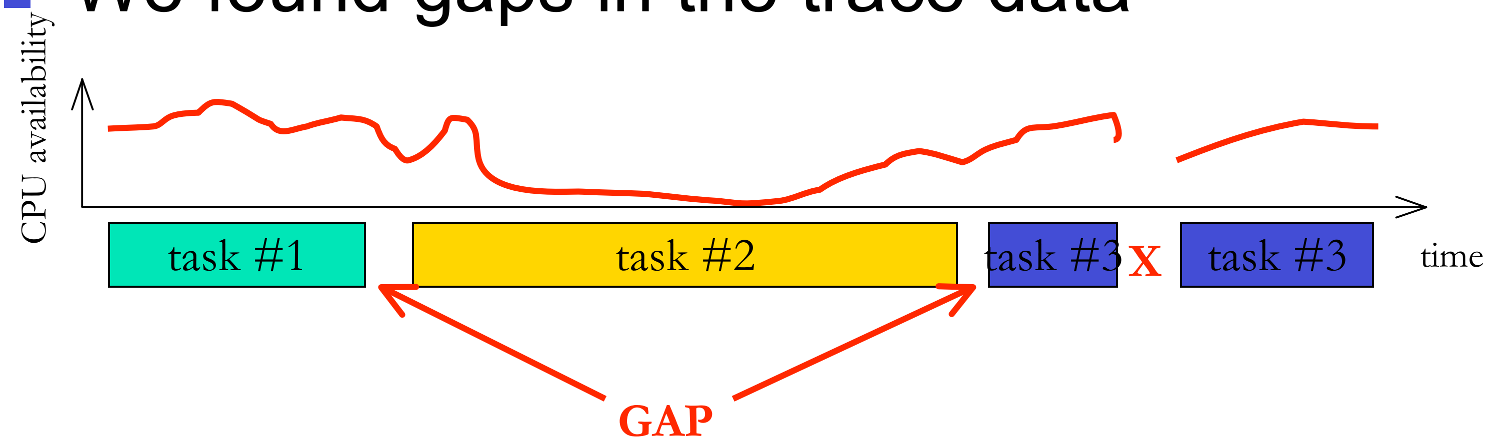
# Host Clock Rates

- First testbed:
  - 230 desktops at the San Diego Supercomputer Center (SDSC) running the Entropia desktop grid software, and 80 desktops at University of Paris-Sud running XtremWeb software
- Obtained traces for 2 months

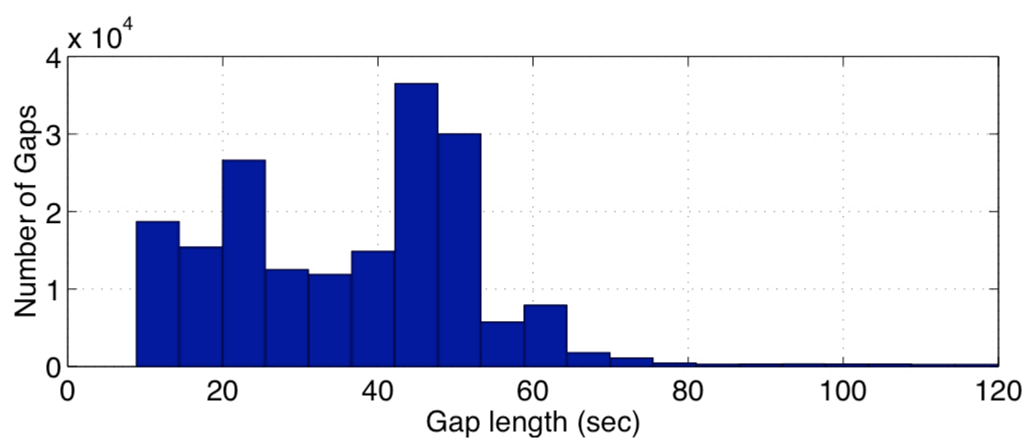


# Cleaning up the data

- We found gaps in the trace data




- Due to server overhead: 35s



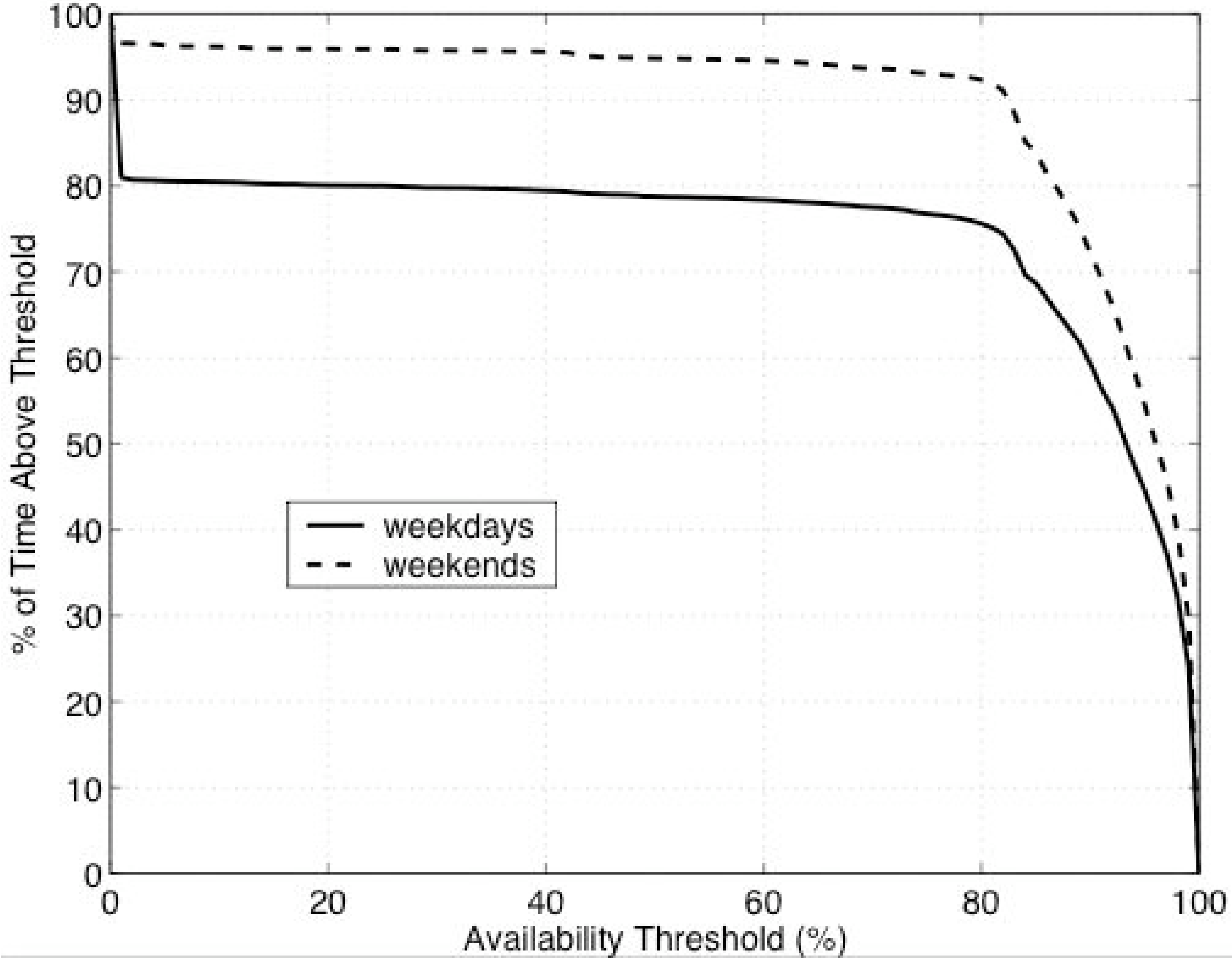
# CPU availability?

- SETI@home uses an all/nothing model
  - If the machine is idle: then use it
  - otherwise: don't use it
- Entropia uses a sophisticated virtual machine
  - monitors machine activity
  - makes sure that the desktop grid application as insignificant interference with the user's job
  - sophisticated but...

# Resource/Task Management

- What happens if a resource gets “reclaimed”?
  - suspend the task and wait?
    - but this may last a long time
  - kill immediately?
    - but then restart from scratch (unless migration is possible?)
    - and perhaps the interruption is only short-lived
- Entropia (and other similar systems) for X minutes, and then give up and kill the task
  - No checkpointing

# An Interesting Results




over all hosts

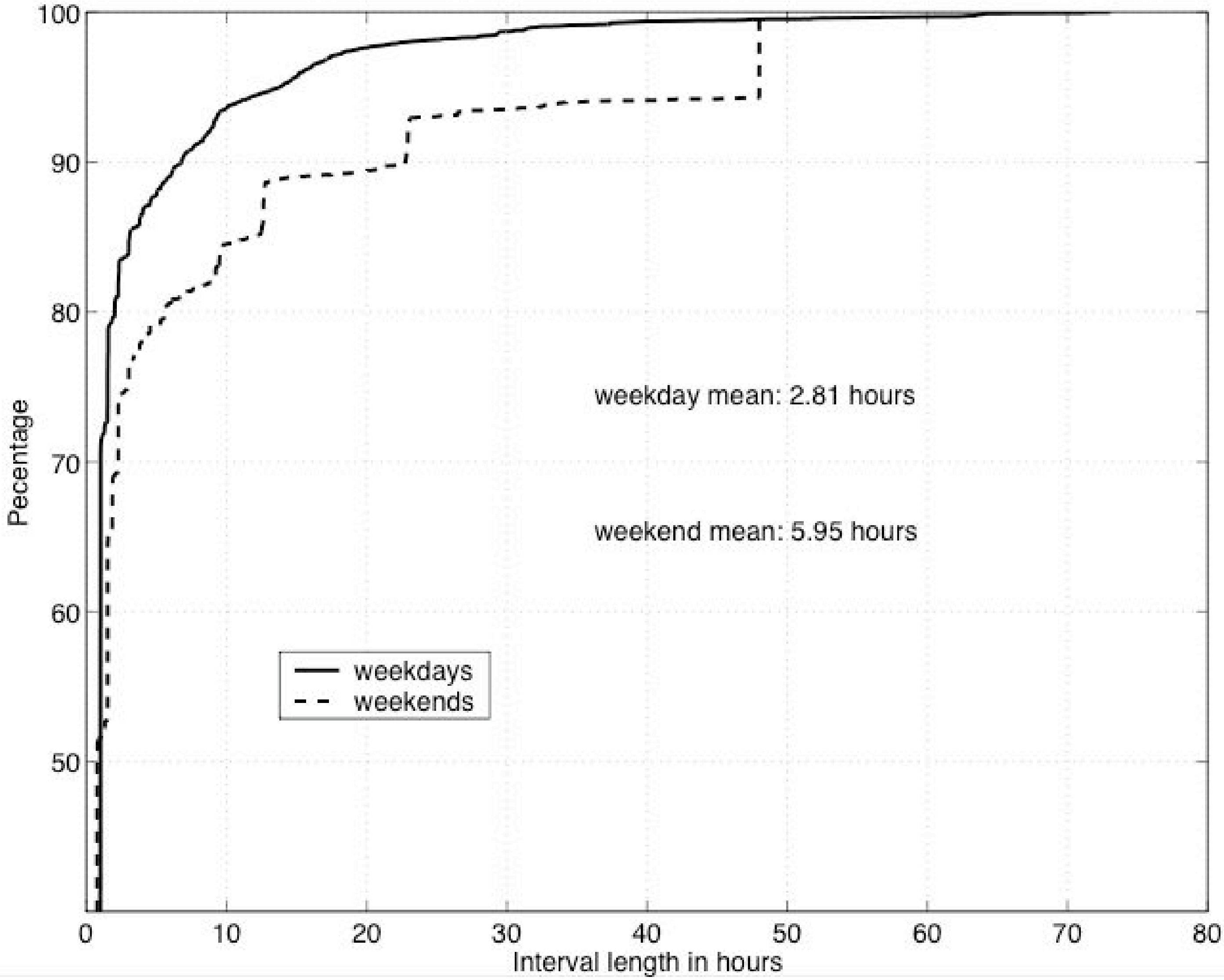
# Conclusion from the graph

- Most machines are either totally busy or more than 80% available
- Therefore one may wonder why it’s so important to have a fancy virtual machine...
- Of course there are different trends between weekends and weekdays

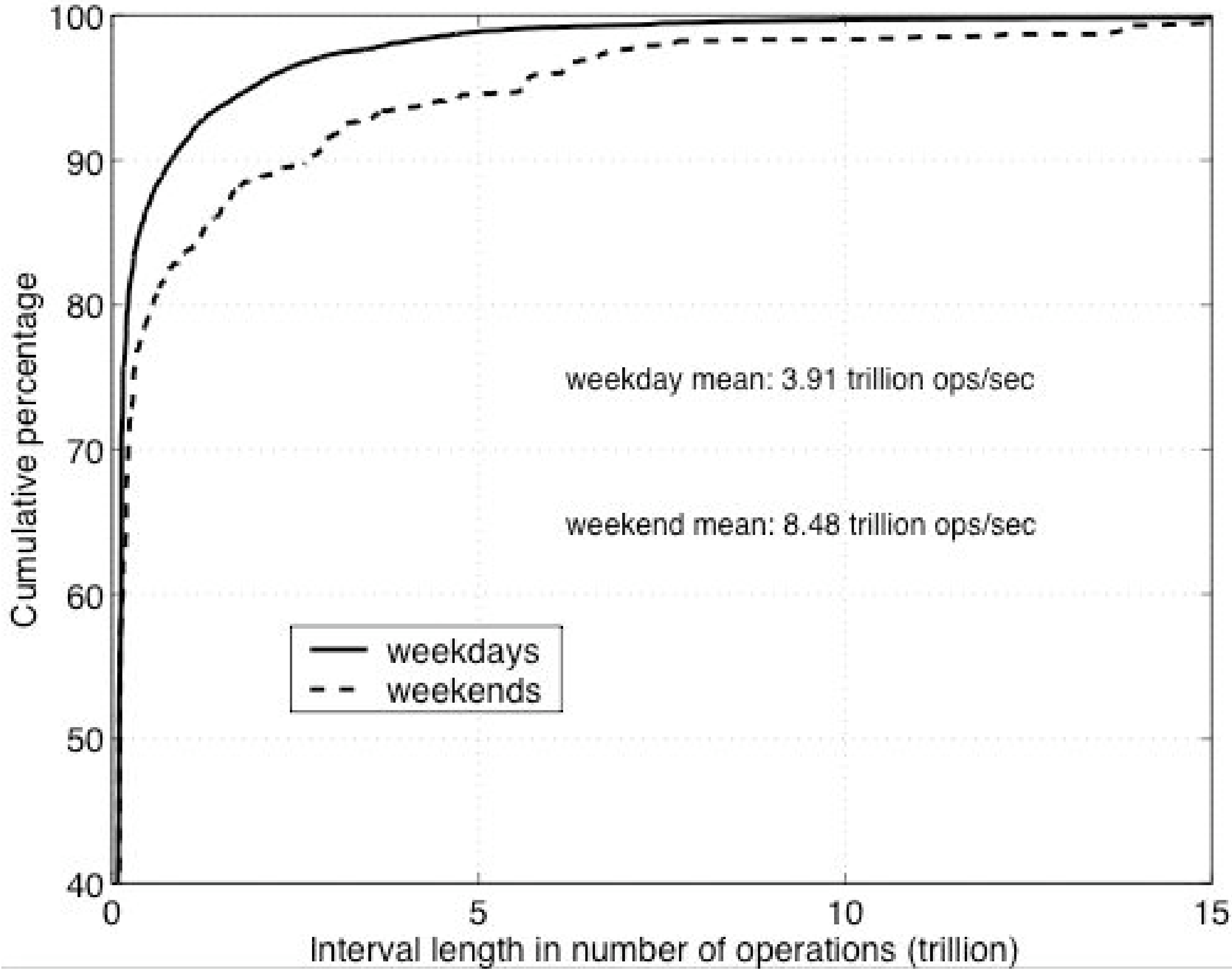
# Availability Intervals

- From the trace data we can compute “availability intervals”
  - Intervals of time during which a task can complete successfully
  - The task may be suspended multiple times during that interval
- We can compute:
  - interval duration in seconds
  - interval duration in terms of number of operations performed

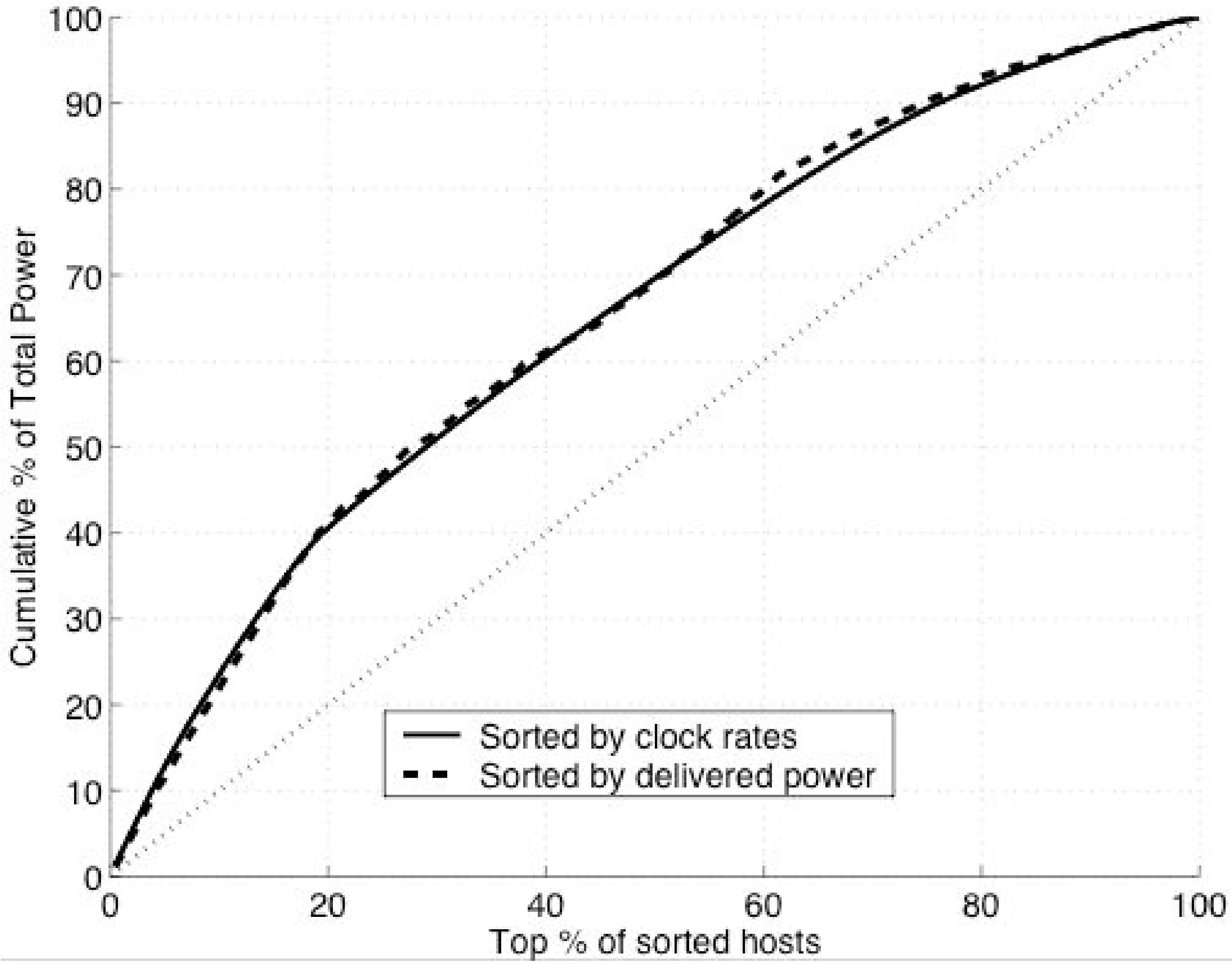
# Availability intervals (sec)



# Availability intervals (ops)



# Total Compute Power



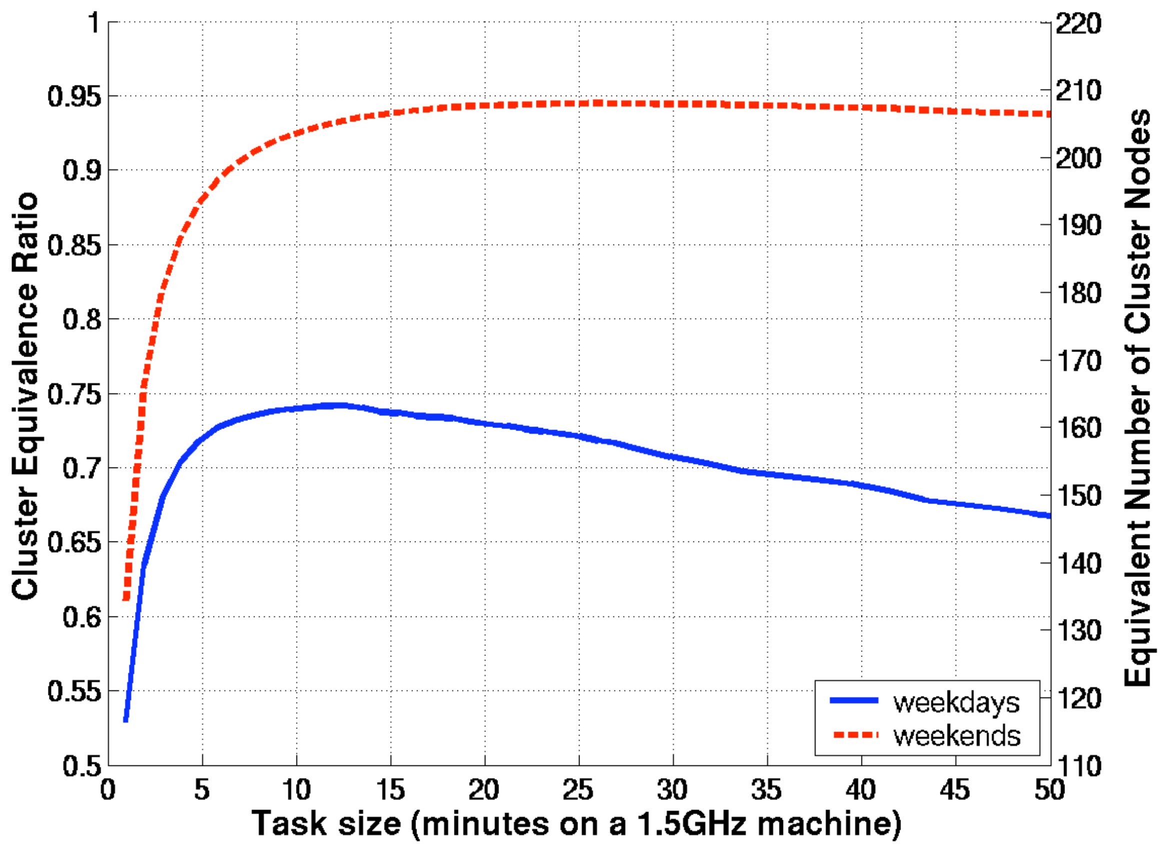
# Conclusion from the graph

- Even slow resources are still useful
  - perhaps there are not as busy because people don’t want to use them?

- Other interesting things
  - how about correlation of availability
  - important for scheduling applications

# Cluster Equivalence

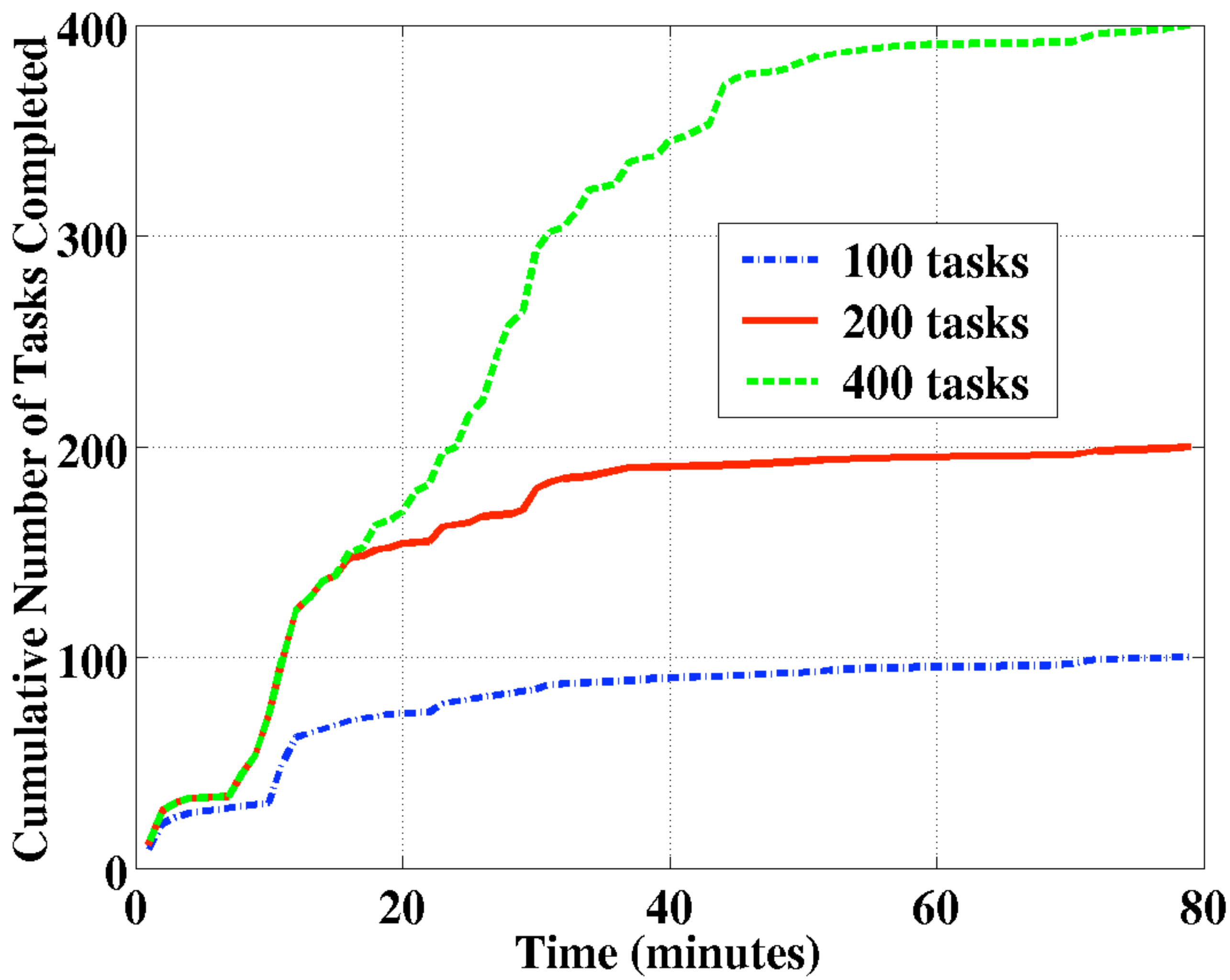
- Cluster of X-nodes with the median compute speed
- Equivalence vs. Task size



# So where are we now?

- Message from the previous results (if we assume it generalizes):
  - If I have an embarrassingly parallel applications
  - If the only thing I care about is throughput
  - If I have a 200 node desktop grid
  - If I can tune the task size
  - I can have the illusion of a 150-node cluster with clock rates at the median of the hosts in the desktop grid
  - Better on weekends
- So this is great, but on a cluster one can do MANY mode things than on a desktop grids
  - i.e., run non-embarrassingly parallel, high-throughput applications
- Question: Could we run less ideal applications effectively?
  - Maybe I only have a few tasks
  - Maybe these tasks communicate

# Fewer tasks than hosts



# Fewer tasks than hosts

- When the number of tasks is small, and when one cares about makespan, the performance of a desktop grid is disappointing
  - Long waiting time for the last task
- The problem is that the issue of resource selection arises
  - Not all hosts are useful
  - All of a sudden the desktop grid must be more complicated
    - Get information about what the hosts are about
    - Use that information to select “good” ones

# Resource Selection Techniques

- Resource Prioritization
  - When I have a choice of multiple hosts, I pick the one with
    - the highest clock rate
    - the one that delivered the most CPU cycles in the past X hours
    - the one that has been the available the longest
    - ...
- Resource Exclusion
  - I  decide never to use hosts with clock rates below X
  - I decide never to use hosts that haven't delivered more than X CPU cycles to the desktop grid in the last Y hours
  - ...
- Task Duplication
  - I send each task to X hosts
    - wasteful if done too much
    - but effective to deal with the "wait for the last task" problem

# Some Results

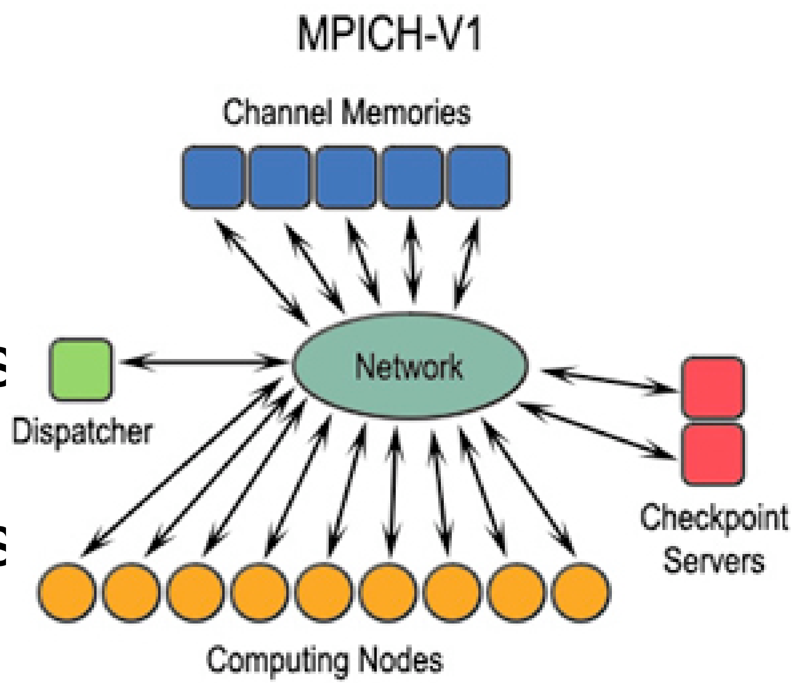
- Researchers have investigated these possibilities (using simulation)
- Some results
  - Prioritization by clock rate works great
    - past history may not be too useful!
  - Resource exclusion by clock rate work ok but not consistently over desktop grids
    - depends too much on the distribution of clock rates
  - Resource exclusion by use of an "artificial deadline" works better but is may be thrown off by one or two very poor predictions
  - Task redundancy is key to deal with poor predictions: twofold replication seems fine
  - By combining all of the above, empirically one can get below a factor 2 of the optimal (assuming a prescient scheduler)
    - And a factor ~3 better than a naive FCFS approach
- Requires improvements to desktop grid infrastructure software

# Running MPI on a Desktop Grid?

- To take things further, and to truly replace a cluster by a desktop grid, one needs to run MPI on volatile nodes
  - Clearly not good for all applications
  - But if the goal is to aggregate memory, perhaps performance is not so critical
- Clearly checkpointing must be used
- Main question: what happens to messages when a node goes down
  - either because of faults
  - or because it is reclaimed
- Note that this is a big issue on large clusters anyway
  - The probability of node failure is high

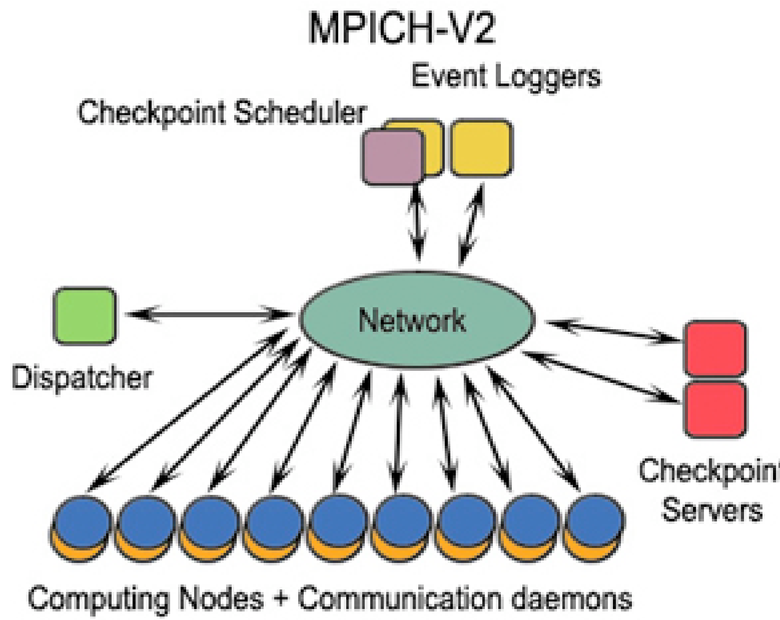
# The MPICH-V Project

- One idea: Use addition processes to store all communication information
  - Message sources/destinations
  - Message sequences
  - Message payload
- Problem:
  - These processes mus be up
  - These processes mus use resources



# The MPICH-V Project

- Some protocols can work without these additional processes
- Idea: relay on “replaying” processes so that messages are “resent”
  - quite complicated



# Conclusion

- Desktop grids are interesting platforms
- Few companies have made a living out of them
  - Many companies have made a living out of clusters
- Researchers are pushing them to do more than they've done in the past
- The future is uncertain
  - Only thin clients and no real exploitable power in the desktop?