# The Quest for Scalable Support of Data Intensive Applications in Distributed Systems


**Ioan Raicu**

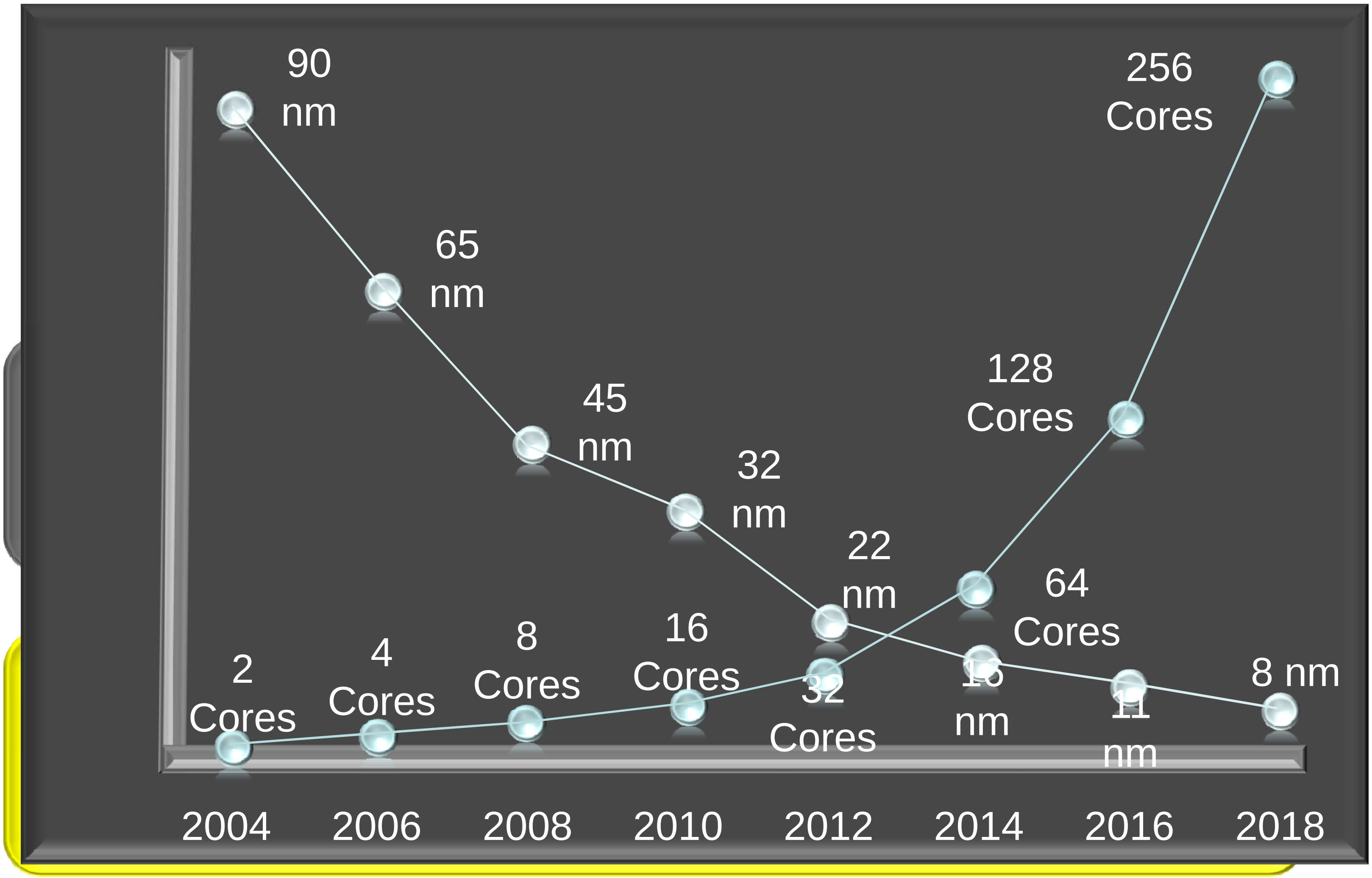
Distributed Systems Laboratory
Computer Science Department
University of Chicago

**In Collaboration with:**
**Ian Foster**, University of Chicago and Argonne National Laboratory
**Alex Szalay**, The Johns Hopkins University
**Yong Zhao**, Microsoft Corporation
**Philip Little, Christopher Moretti, Amitabh Chaudhary, Douglas Thain**, University of Notre Dame

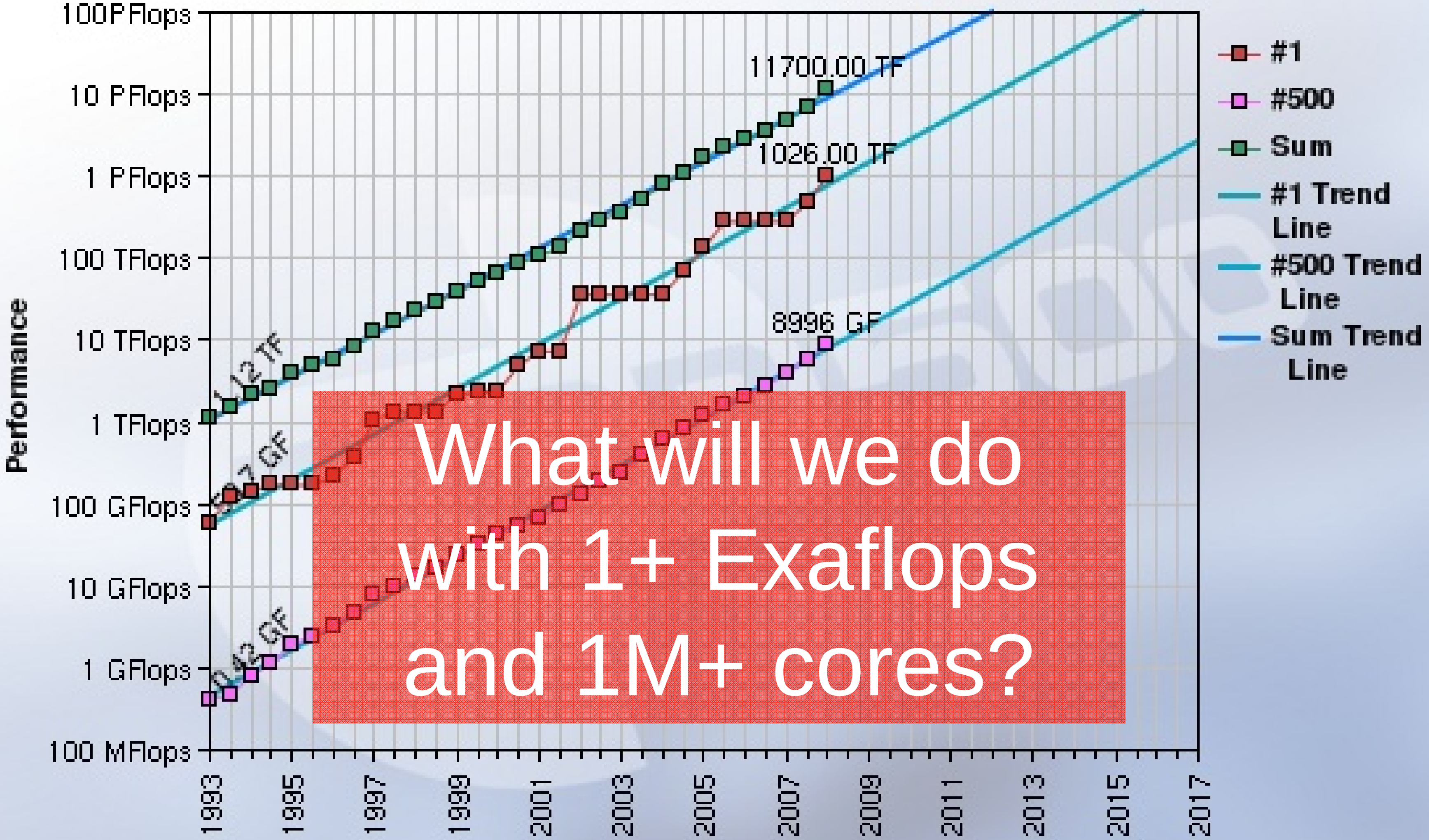


IEEE/ACM Supercomputing 2008
Argonne National Laboratory Booth
November 20th, 2008

# Many-Core Growth Rates

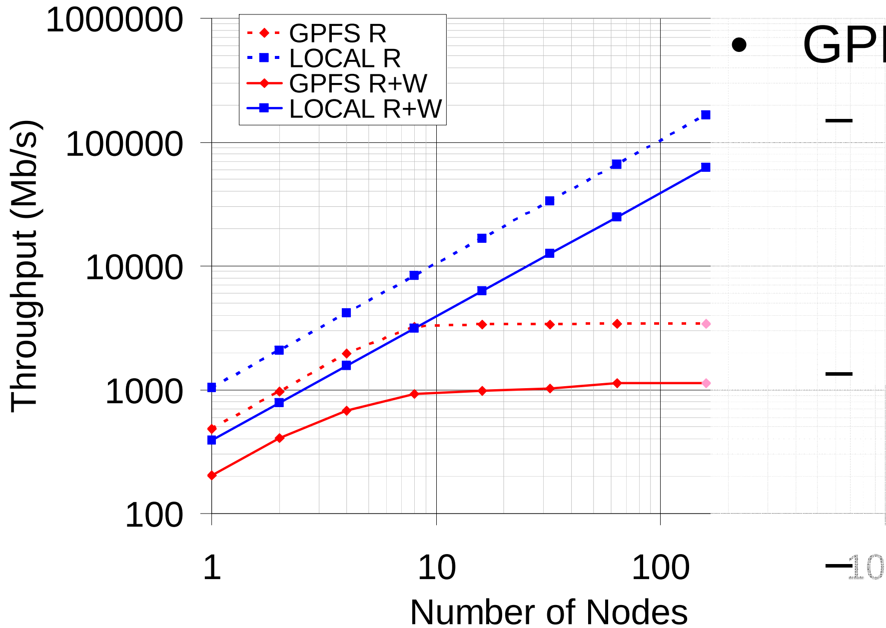


# Projected Performance Development




What will we do with 1+ Exaflops and 1M+ cores?

# Storage Resource Scalability




- GPFS vs. LOCAL
  - Read Throughput
    - 1 node: 0.48Gb/s vs. 1.03Gb/s ➔ **2.15x**
    - 160 nodes: 3.4Gb/s vs. 165Gb/s ➔ **48x**
    - **11Mb/s per CPU vs. 515Mb/s per CPU**
  - Read+Write Throughput:
    - 1 node: 0.2Gb/s vs. 0.39Gb/s ➔ **1.95x**
    - 160 nodes: 1.1Gb/s vs. 62Gb/s ➔ **55x**
  - Metadata (mkdir / rm -rf)
    - 1 node: 151/sec vs. 199/sec ➔ **1.3x**
    - 160 nodes: 21/sec vs. 31840/sec ➔ **1516x**
- IBM BlueGene/P
  - 160K CPU cores
  - GPFS 8GB/s I/O rates (16 servers)
  - Experiments on 160K CPU BG/P achieved 0.3Mb/s per CPU core
  - Experiments on 5.7K CPU SiCortex achieved 0.06Mb/s per CPU core

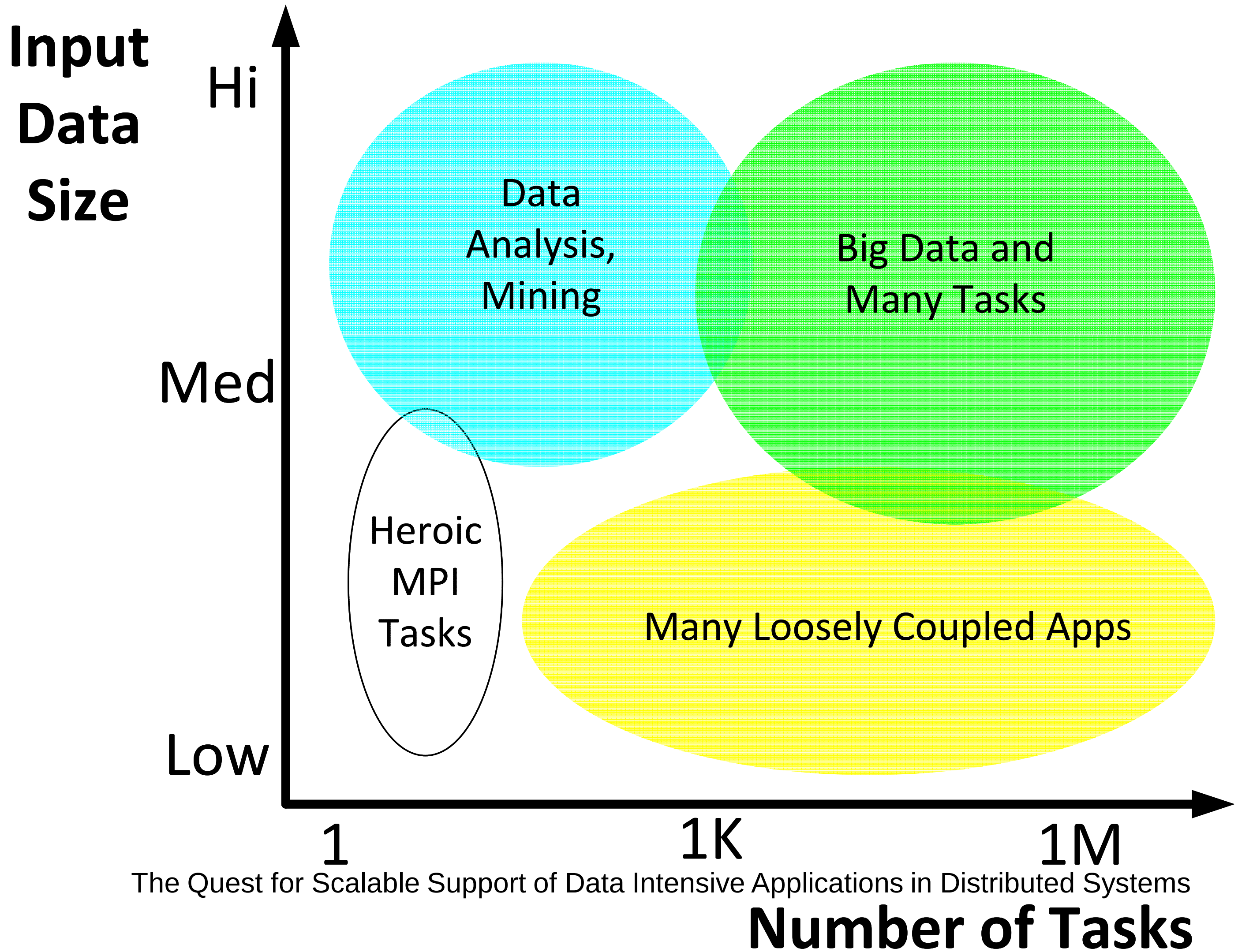
# Programming Model Issues

- **Multicore** processors
- Massive **task parallelism**
- Massive **data parallelism**
- Integrating **black box applications**
- Complex **task dependencies** (task graphs)
- **Failure**, and other execution management issues
- **Dynamic task graphs**
- Documenting **provenance** of data products
- **Data management**: input, intermediate, output
- **Dynamic data access** involving large amounts of data

# Programming Model Issues

- **Multicore** processors
- Massive **task parallelism**
- Massive **data parallelism**
- Integrating **black box applications**
- Complex **task dependencies** (task graphs)
- **Failure**, and other execution management issues
- **Dynamic task graphs**
- Documenting **provenance** of data products
- **Data management**: input, intermediate, output
- **Dynamic data access** involving large amounts of data

# Problem Types



# An Incomplete and Simplistic View of Programming Models and Tools

**Single task, modest data**
MPI, etc…

**Many Tasks**
DAGMan+Pegasus
Karajan+Swift+Falkon

**Much Data**
MapReduce/Hadoop
Dryad

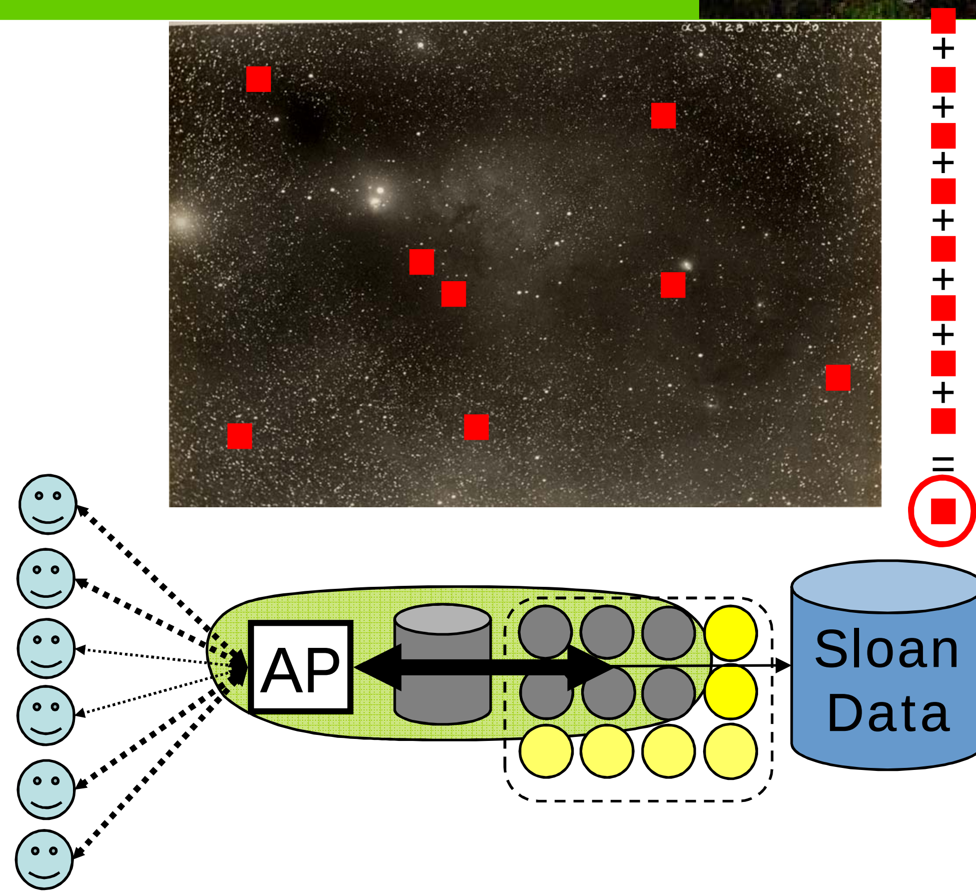
**Complex Tasks, Much Data**
Dryad, Pig, Sawzall
Swift+Falkon (using data diffusion)

# MTC: Many Task Computing

- Loosely coupled applications
  - High-performance computations comprising of multiple distinct activities, coupled via file system operations or message passing
  - Emphasis on using many resources over short time periods
  - Tasks can be:
    - small or large, independent and dependent, uniprocessor or multiprocessor, compute-intensive or data-intensive, static or dynamic, homogeneous or heterogeneous, loosely or tightly coupled, large number of tasks, large quantity of computing, and large volumes of data…

# Motivating Example: AstroPortal Stacking Service

- Purpose
  - On-demand “stacks” of random locations within ~10TB dataset
- Challenge
  - Processing Costs:
    - O(100ms) per object
  - Data Intensive:
    - 40MB:1sec
  - Rapid access to 10-10K “random” files
  - Time-varying load



# Hypothesis

*"Significant performance improvements can be obtained in the analysis of large dataset by leveraging information about data analysis workloads rather than individual data analysis tasks."*

- **Important concepts related to the hypothesis**
  - **Workload**: a complex query (or set of queries) decomposable into simpler tasks to answer broader analysis questions
  - **Data locality** is crucial to the efficient use of large scale distributed systems for scientific and data-intensive applications
  - Allocate computational and caching storage resources, **co-scheduled** to optimize workload performance

# Abstract Model

- AMDASK: An Abstract Model for DAta-centric taSK farms
  - Task Farm: A common parallel pattern that drives independent computational tasks
- Models the efficiency of data analysis workloads for the MTC class of applications
- Captures the following data diffusion properties
  - Resources are acquired in response to demand
  - Data and applications diffuse from archival storage to new resources
  - Resource “caching” allows faster responses to subsequent requests
  - Resources are released when demand drops
  - Considers both data and computations to optimize performance

# AMDASK: Base Definitions

- **Data Stores:** Persistent & Transient
  – Store capacity, load, ideal bandwidth, available bandwidth
- **Data Objects:**
  – Data object size, *data object's storage location(s),* copy time
- **Transient resources:** compute speed, resource state
- **Task:** application, input/output data

# AMDASK: Execution Model Concepts

- Dispatch Policy
  - next-available, first-available, max-compute-util, max-cache-hit
- Caching Policy
  - random, FIFO, LRU, LFU
- Replay policy
- Data Fetch Policy
  - Just-in-Time, Spatial Locality
- Resource Acquisition Policy
  - one-at-a-time, additive, exponential, all-at-once, optimal
- Resource Release Policy
  - distributed, centralized

# AMDASK: Performance Efficiency Model

- B: Average Task Execution Time:
  - K: Stream of tasks
  - μ(k): Task k execution time

$$B = \frac{1}{|K|}\sum_{k \in K} \mu(\kappa)$$

- Y: Average Task Execution Time with Overheads:
  - o(k): Dispatch overhead
  - ς(δ,τ): Time to get data

$$Y = \begin{cases} \frac{1}{|K|}\sum_{\kappa \in K}[\mu(\kappa)+o(\kappa)], & \delta \in \phi(\tau), \delta \in \Omega \\ \frac{1}{|K|}\sum_{\kappa \in K}[\mu(\kappa)+o(\kappa)+\zeta(\delta,\tau)], & \delta \notin \phi(\tau), \delta \in \Omega \end{cases}$$

- V: Workload Execution Time:
  - A: Arrival rate of tasks
  - T: Transient Resources

$$V = \max\left(\frac{B}{|T|}, \frac{1}{A}\right) * |K|$$

- W: Workload Execution Time with Overheads

$$W = \max\left(\frac{Y}{|T|}, \frac{1}{A}\right) * |K|$$

# AMDASK: Performance Efficiency Model

- **Efficiency**

$$\mathrm{E} = \frac{V}{W} \longrightarrow E = \begin{cases} 1, & \dfrac{Y}{|T|} \leq \dfrac{1}{A} \\ \max\left(\dfrac{B}{Y}, \dfrac{|\mathrm{T}|}{\mathrm{A}*Y}\right), & \dfrac{Y}{|T|} > \dfrac{1}{A} \end{cases}$$

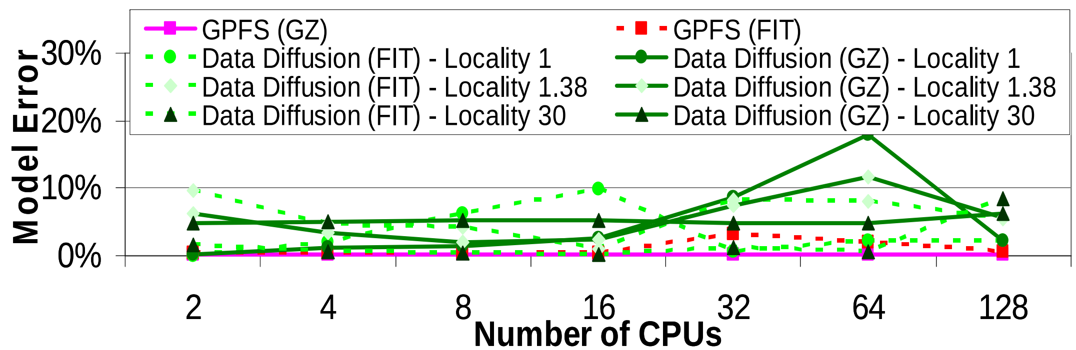
- **Speedup**

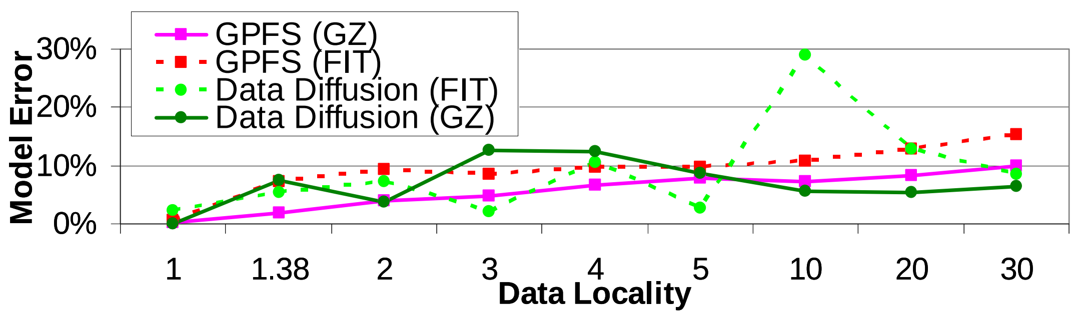
$$S = E * |T|$$

- **Optimizing Efficiency**
  - Easy to maximize either efficiency or speedup independently
  - Harder to maximize both at the same time
    - Find the smallest number of *transient resources* |T| while maximizing speedup*efficiency

# Model Validation

- Stacking service (large scale astronomy application)
- 92 experiments
- 558K files
  - Compressed: 2MB each ➔ 1.1TB
  - Un-compressed: 6MB each ➔ 3.3TB



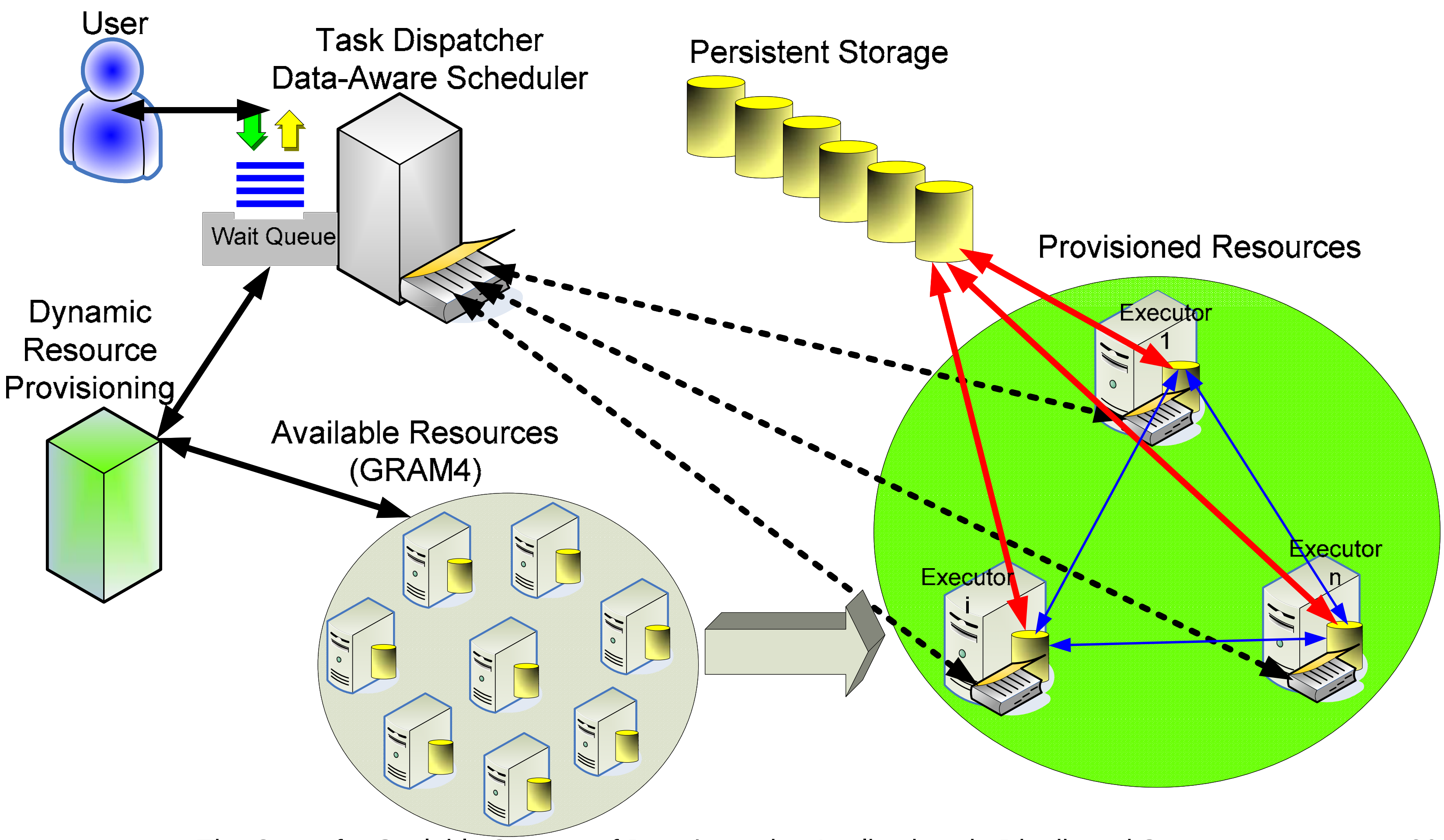
# Falkon: a Fast and Light-weight tasK executiON framework

- ***Goal:*** enable the ***rapid and efficient*** execution of many independent jobs on large compute clusters
- Combines three components:
  - a ***streamlined task dispatcher***
  - ***resource provisioning*** through multi-level scheduling techniques
  - ***data diffusion*** and data-aware scheduling to leverage the co-located computational and storage resources
- Integration into Swift to leverage many applications
  - Applications cover many domains: astronomy, astro-physics, medicine, chemistry, economics, climate modeling, etc

# Falkon: a Fast and Light-weight tasK executiON framework

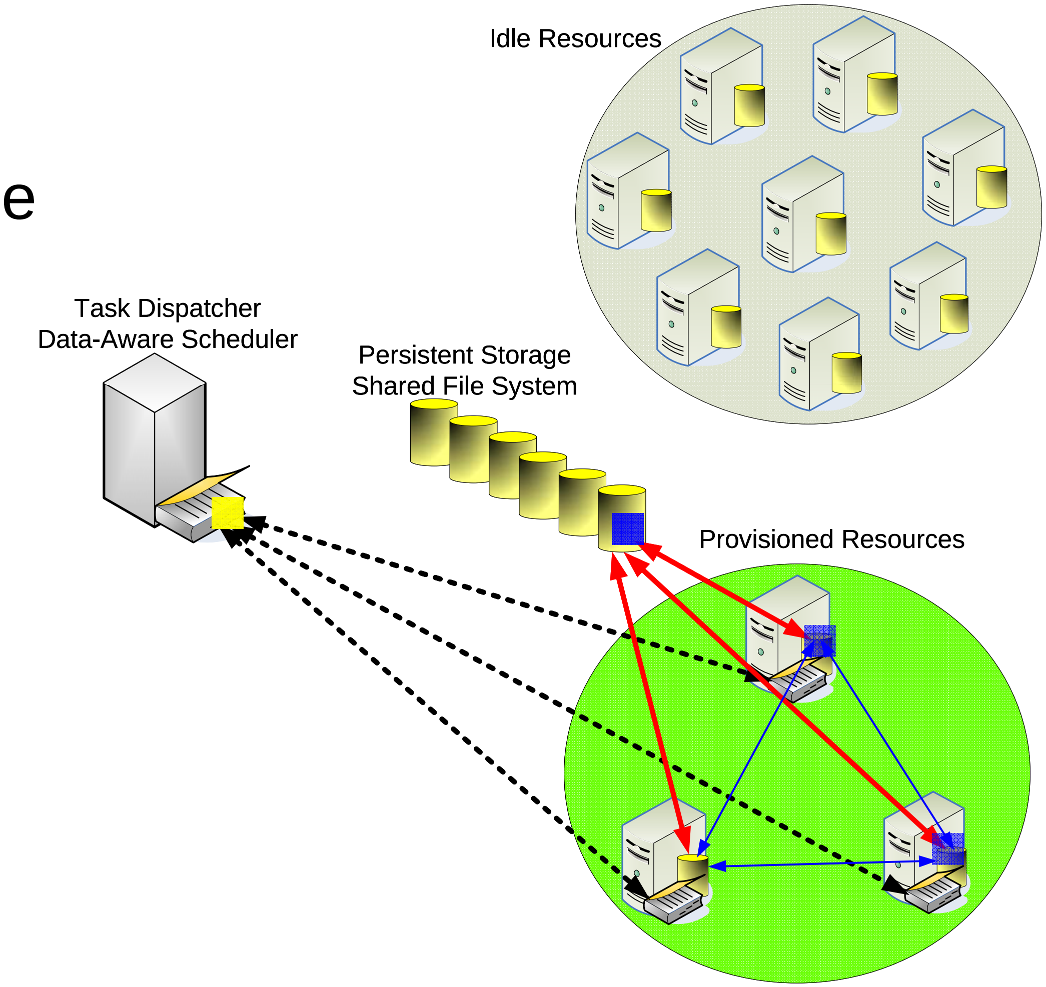
- ***Goal:*** enable the ***rapid and efficient*** execution of many independent jobs on large compute clusters
- Combines three components:
  - a ***streamlined task dispatcher***
  - ***resource provisioning*** through multi-level scheduling techniques
  - ***data diffusion*** and data-aware scheduling to leverage the co-located computational and storage resources
- Integration into Swift to leverage many applications
  - Applications cover many domains: astronomy, astro-physics, medicine, chemistry, economics, climate modeling, etc

# Falkon Overview



# Data Diffusion

- Resource acquired in response to demand
- Data and applications diffuse from archival storage to newly acquired resources
- Resource “caching” allows faster responses to subsequent requests
  - Cache Eviction Strategies: RANDOM, FIFO, LRU, LFU
- Resources are released when demand drops



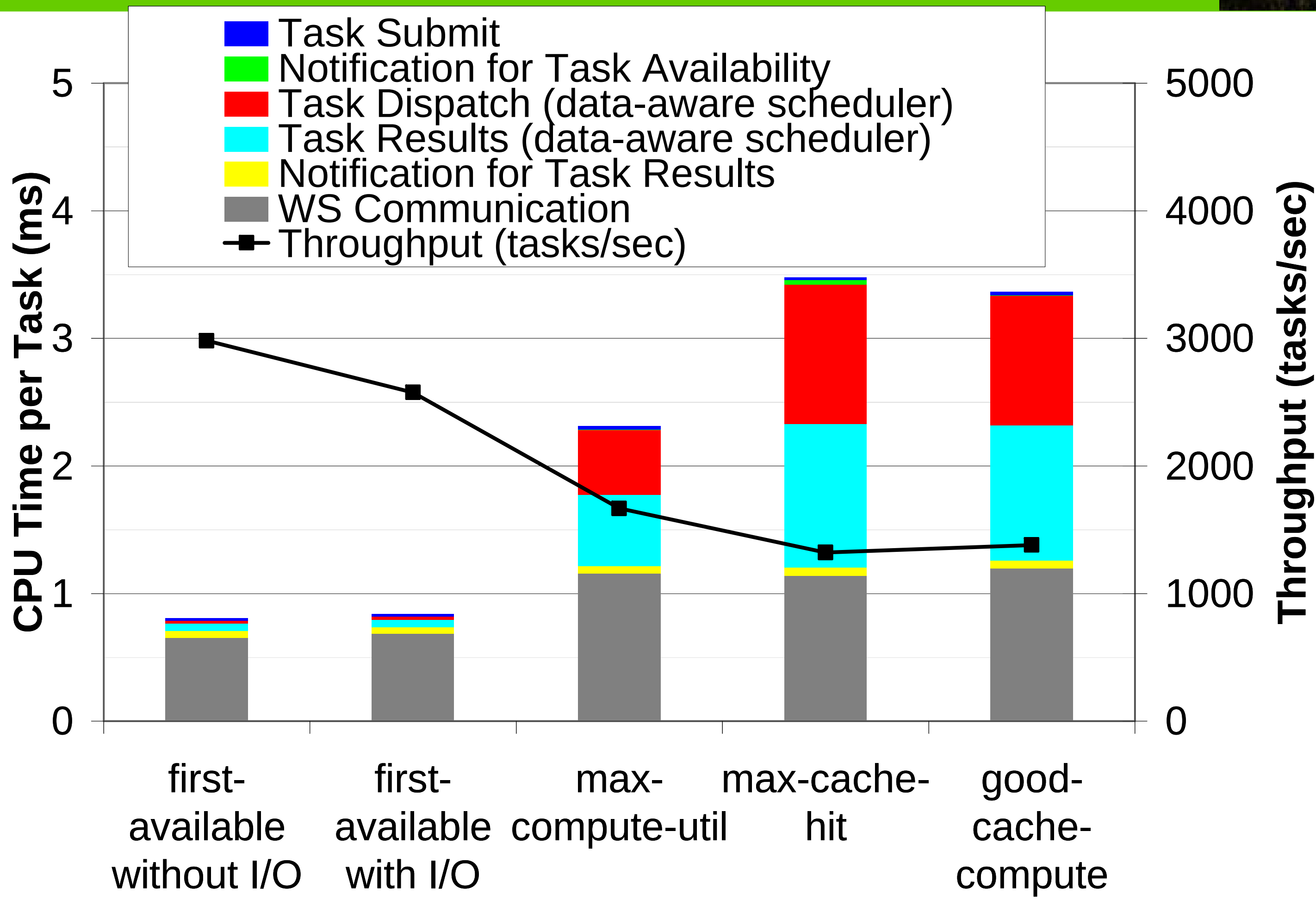
# Data Diffusion

- Considers both data and computations to optimize performance
  - Supports data-aware scheduling
  - Can optimize compute utilization, cache hit performance, or a mixture of the two
- Decrease dependency of a shared file system
  - Theoretical linear scalability with compute resources
  - Significantly increases meta-data creation and/or modification performance
- Central for “data-centric task farm” realization

# Scheduling Policies

- first-available:
  - simple load balancing
- max-cache-hit
  - maximize cache hits
- max-compute-util
  - maximize processor utilization
- good-cache-compute
  - maximize both cache hit and processor utilization at the same time

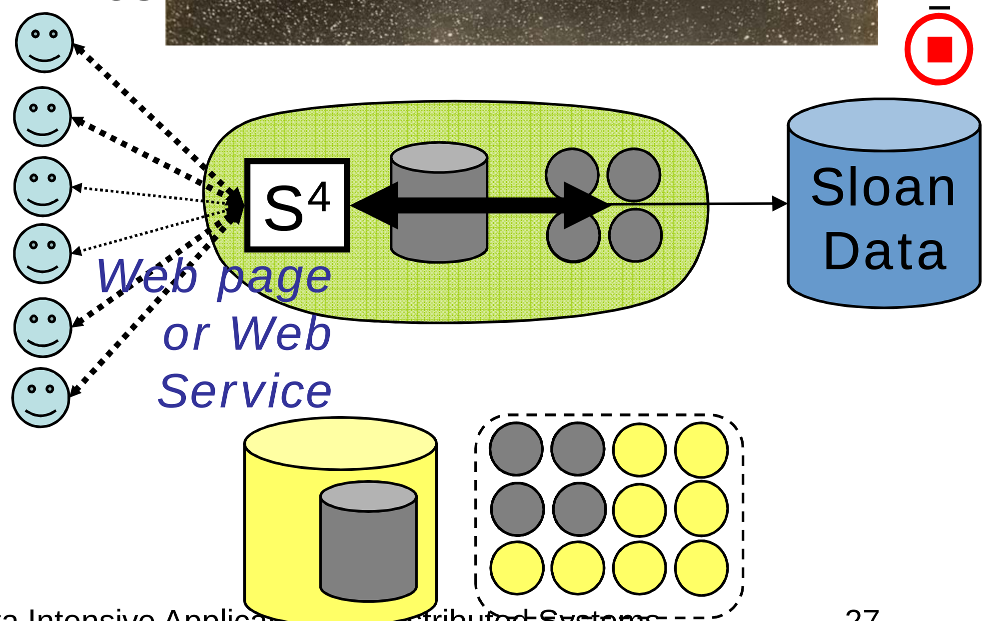
# Data-Aware Scheduler Profiling



# AstroPortal Stacking Service

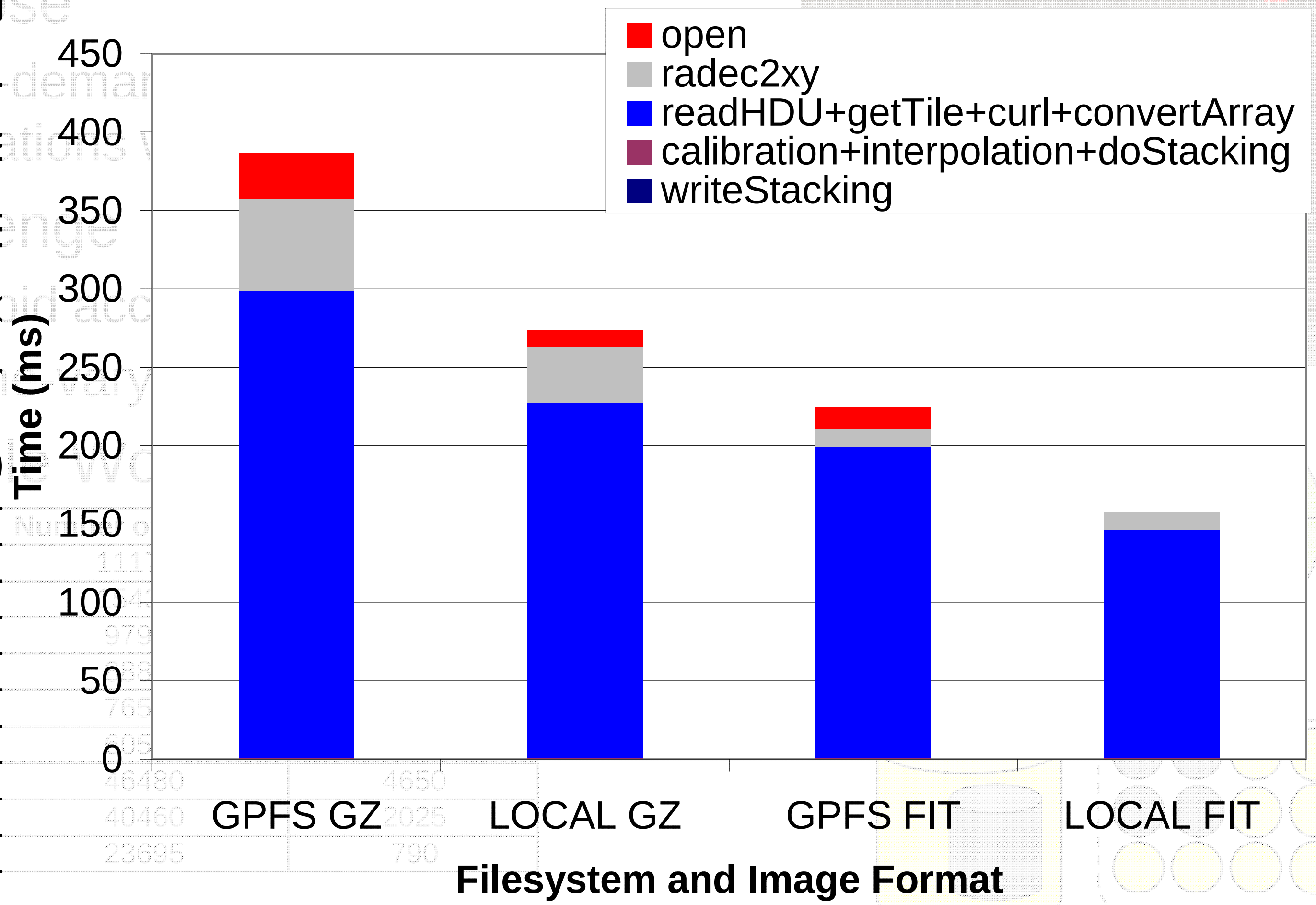
- Purpose
  - On-demand “stacks” of random locations within ~10TB dataset
- Challenge
  - Rapid access to 10-10K “random” files
  - Time-varying load
- Sample Workloads

| Locality | Number of Objects | Number of Files |
|---|---|---|
| 1 | 111700 | 111700 |
| 1.38 | 154345 | 111699 |
| 2 | 97999 | 49000 |
| 3 | 88857 | 29620 |
| 4 | 76575 | 19145 |
| 5 | 60590 | 12120 |
| 10 | 46480 | 4650 |
| 20 | 40460 | 2025 |
| 30 | 23695 | 790 |



# AstroPortal Stacking Service

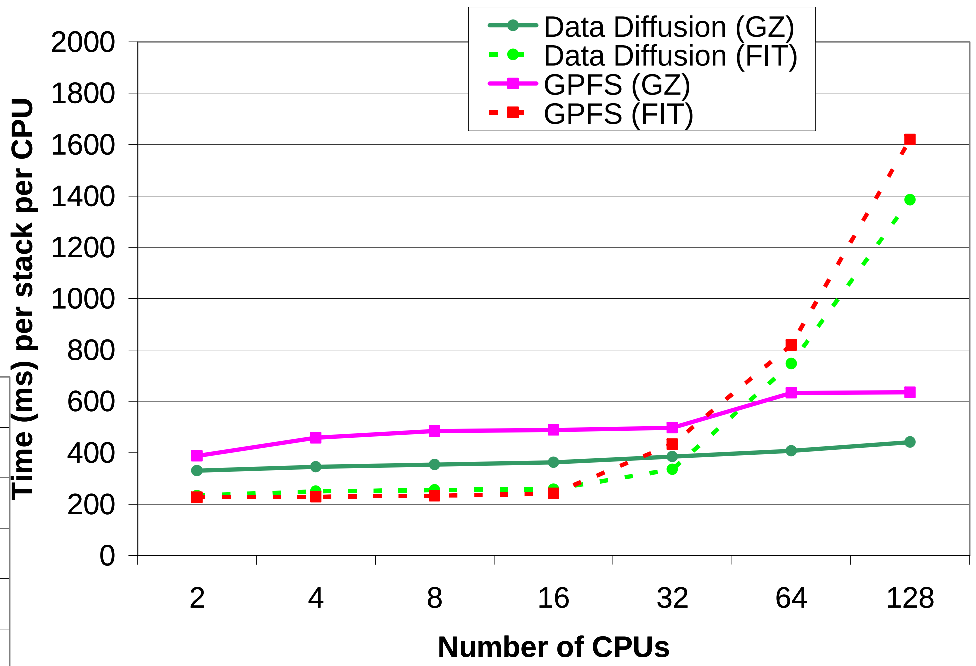

- Purpose
  - On-
    loca
- Challe
  - Ra
  - Tim
- Samp

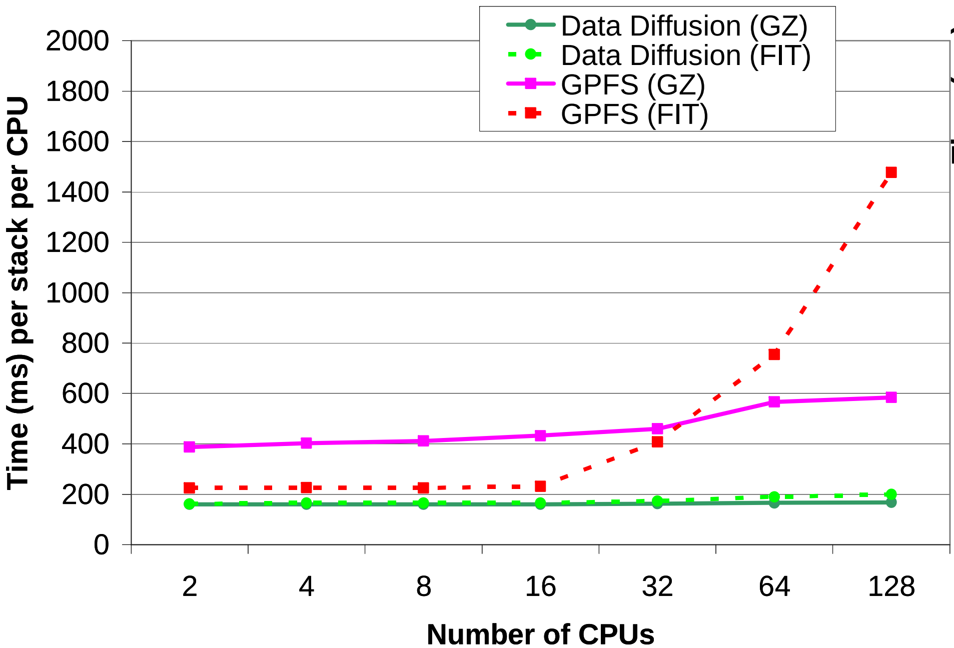



| Locality |
|---|
| 1 |
| 1.38 |
| 2 |
| 3 |
| 4 |
| 5 |
| 10 |
| 20 |
| 30 |

# AstroPortal Stacking Service with Data Diffusion

- Low data locality →
  - Similar (but better) performance to GPFS

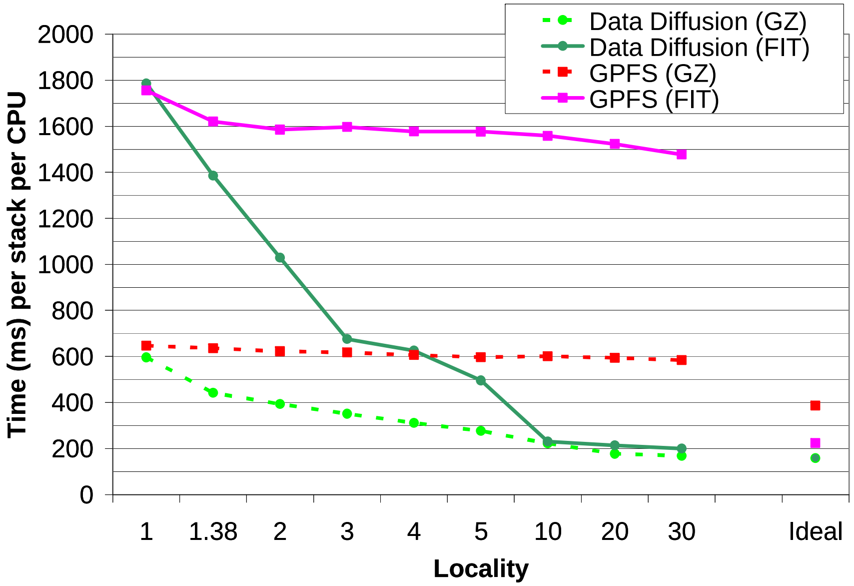


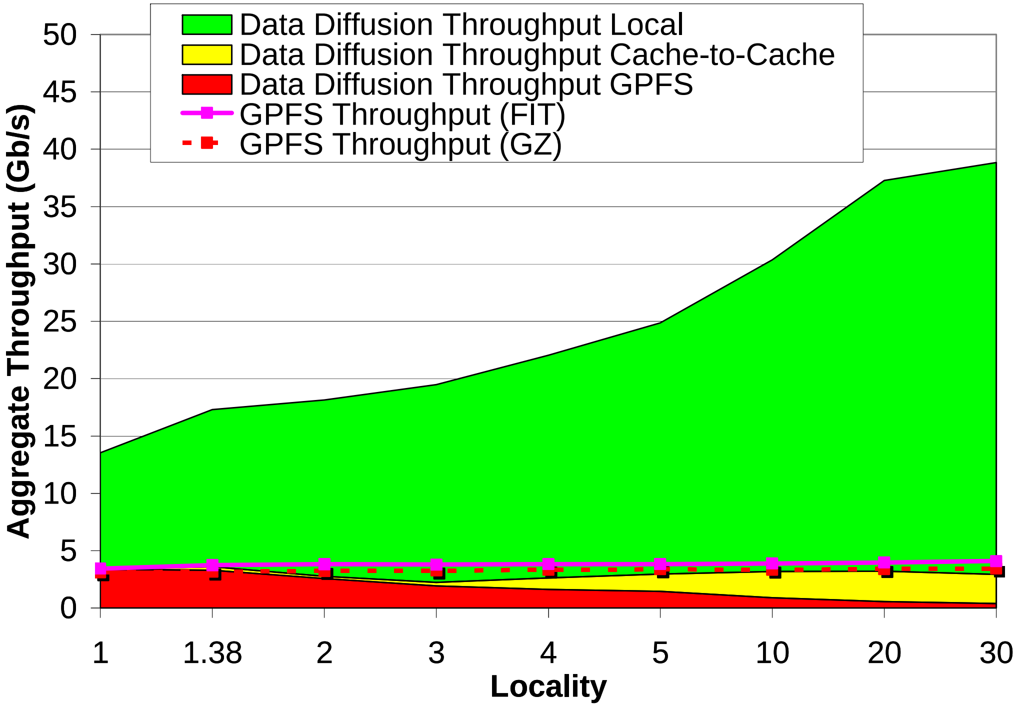




- ← High data locality
  - Near perfect scalability

# AstroPortal Stacking Service with Data Diffusion

- Aggregate throughput:
  - 39Gb/s
  - 10X higher than GPFS
- Reduced load on GPFS
  - 0.49Gb/s
  - 1/10 of the original load

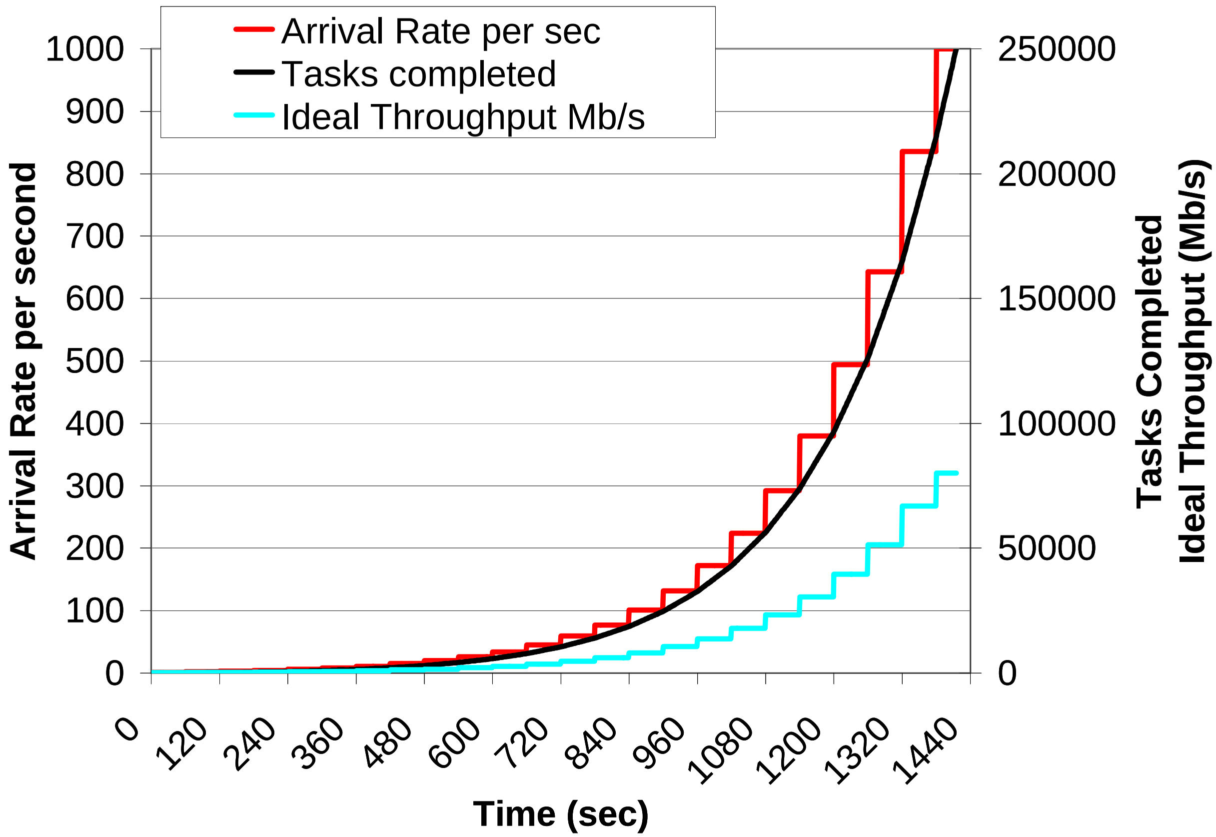



- Big performance gains as locality increases

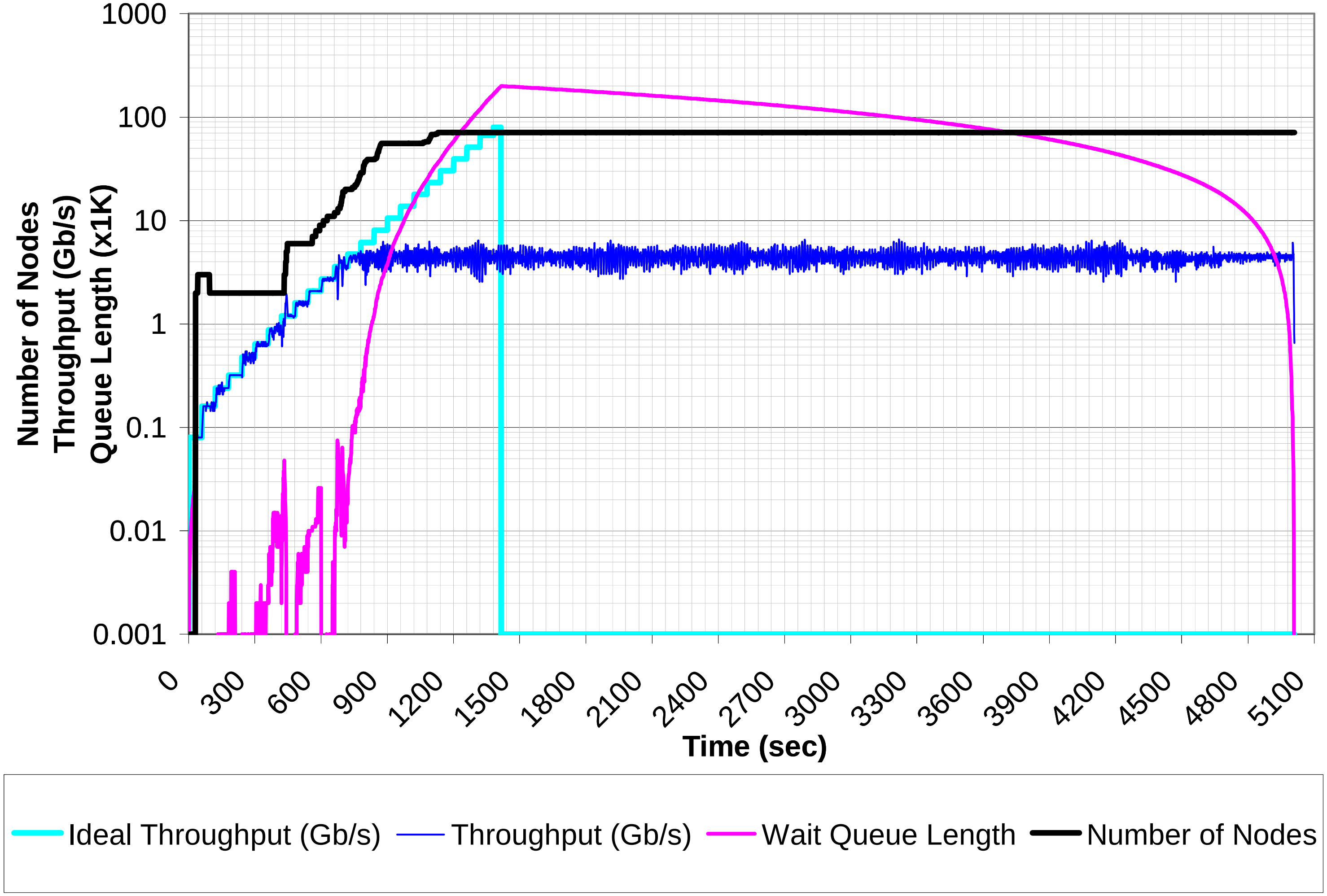
# Monotonically Increasing Workload

- 250K tasks
  - 10MB reads
  - 10ms compute
- Vary arrival rate:
  - Min: 1 task/sec
  - Increment function: CEILING(*1.3)
  - Max: 1000 tasks/sec
- 128 processors
- Ideal case:
  - 1415 sec
  - 80Gb/s peak throughput



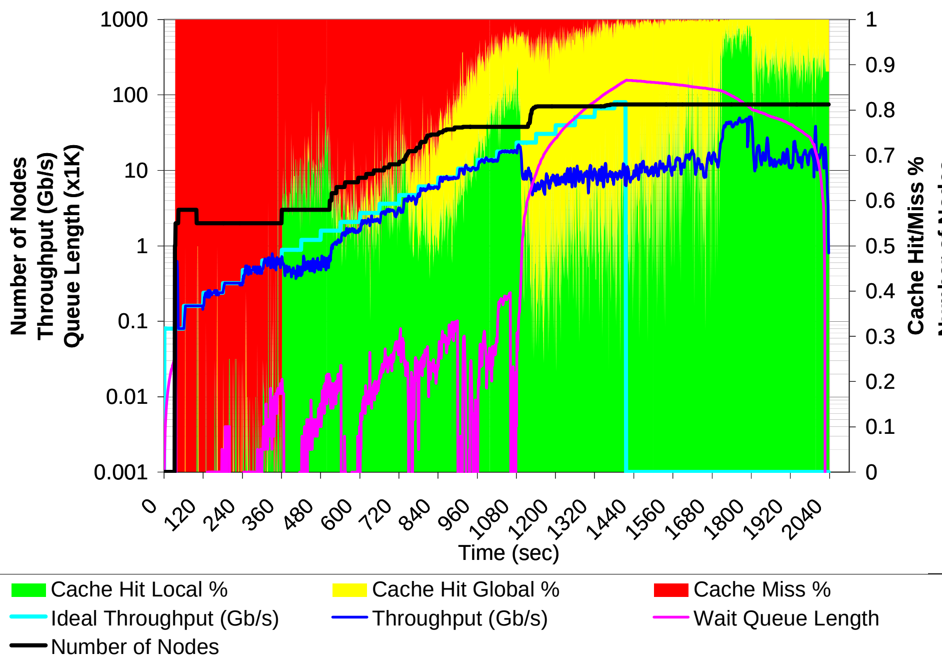
# Data Diffusion: First-available (GPFS)

- **GPFS vs. ideal**: 5011 sec vs. 1415 sec

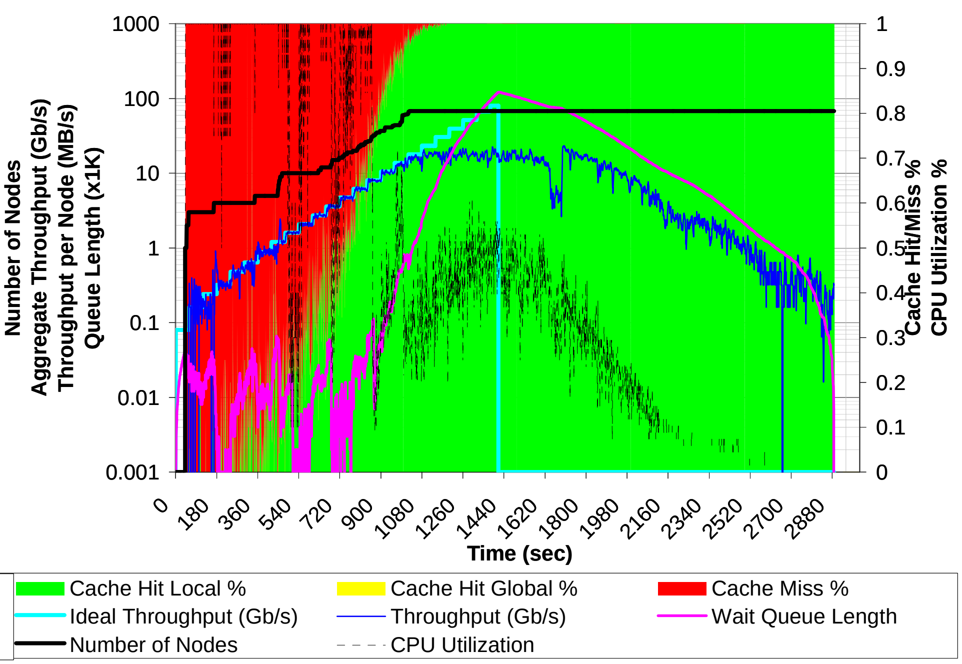


# Data Diffusion: Max-compute-util & max-cache-hit

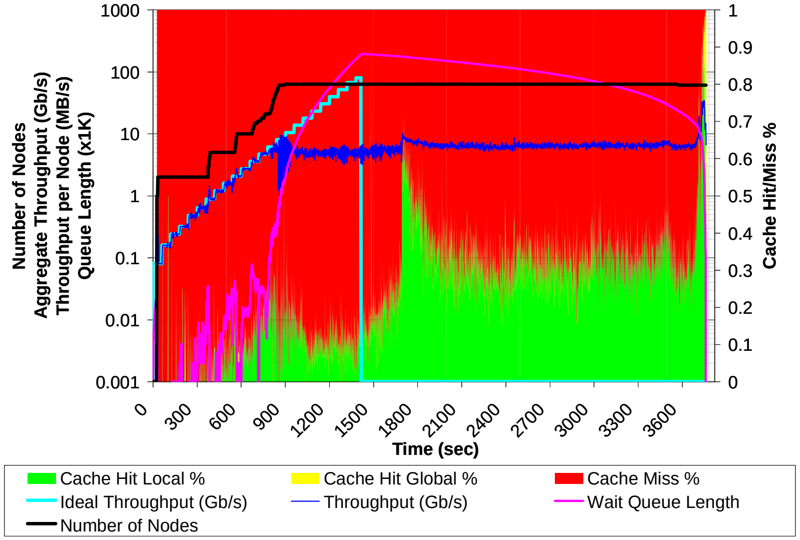
## Max-compute-util




## Max-cache-hit

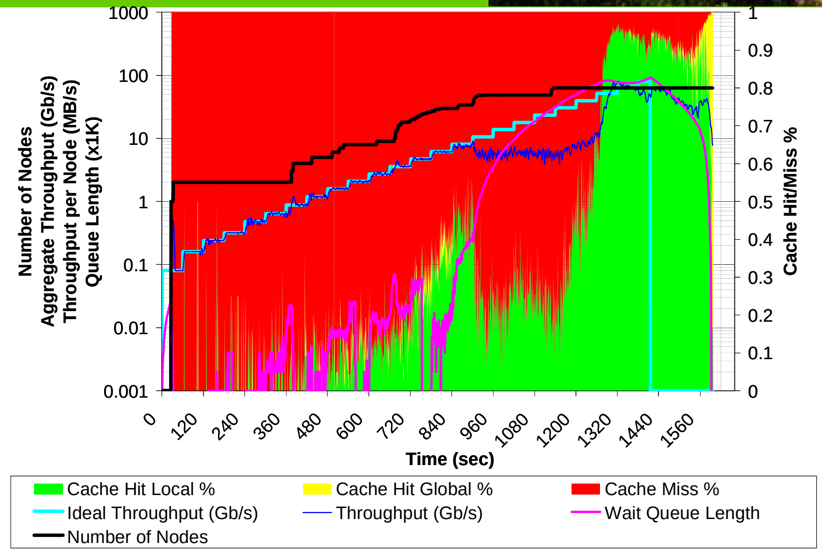


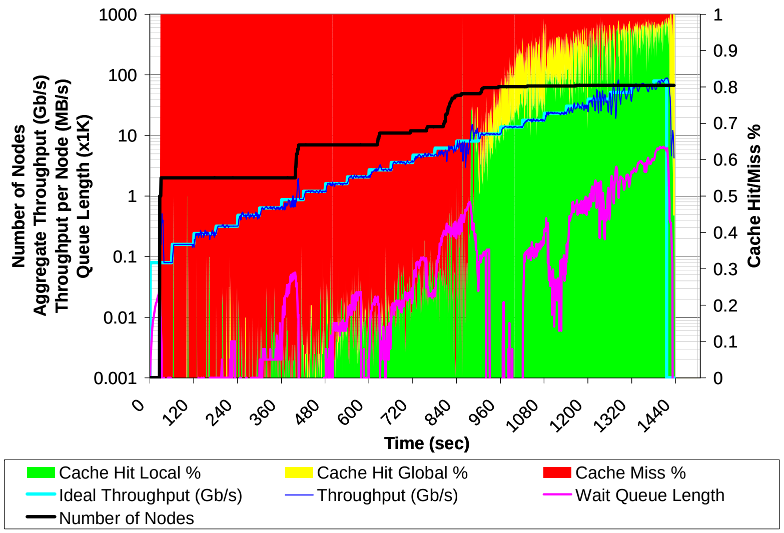
# Data Diffusion: Good-cache-compute




← 1GB

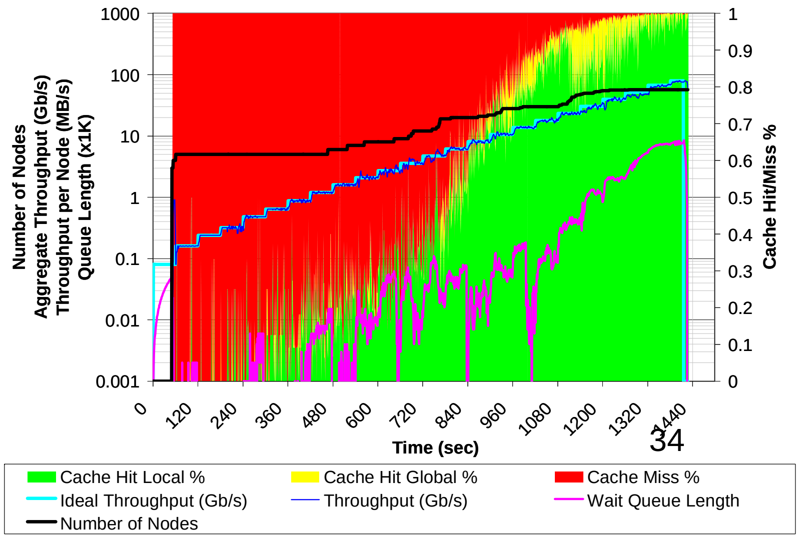
1.5GB →



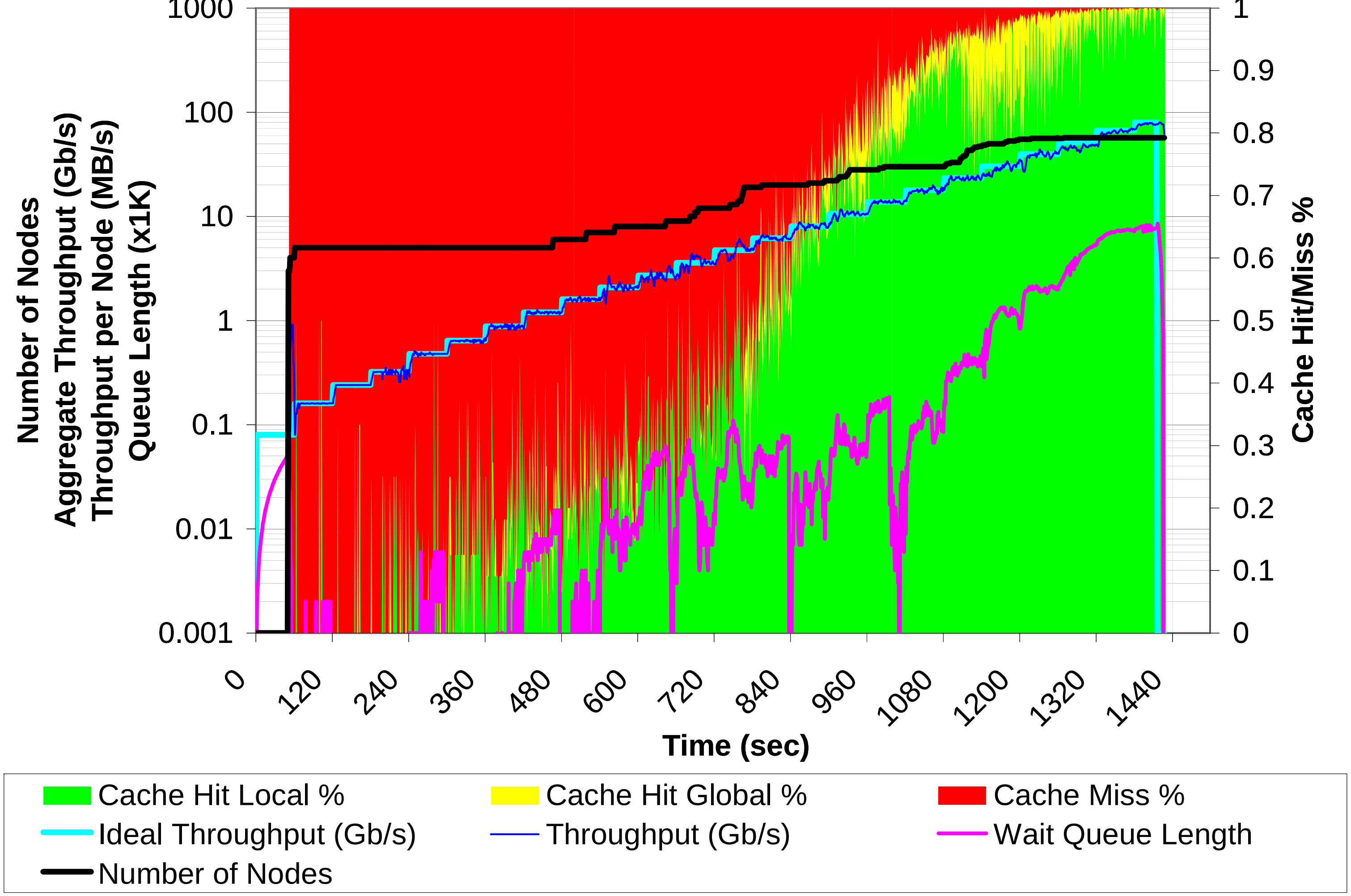

← 2GB

4GB →

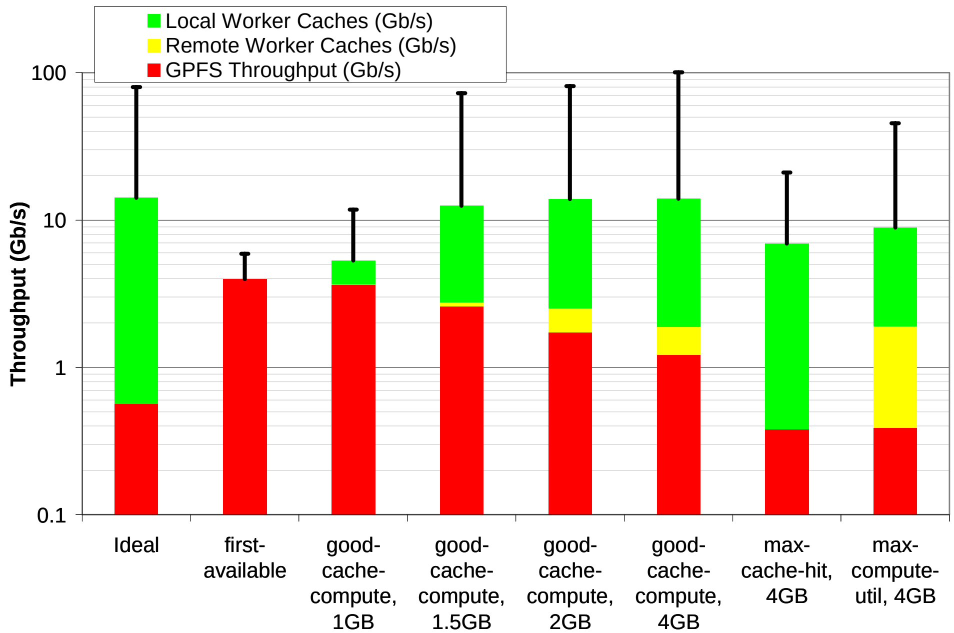


# Data Diffusion: Good-cache-compute

- **Data Diffusion vs. ideal**: 1436 sec vs 1415 sec



# Data Diffusion: Throughput and Response Time

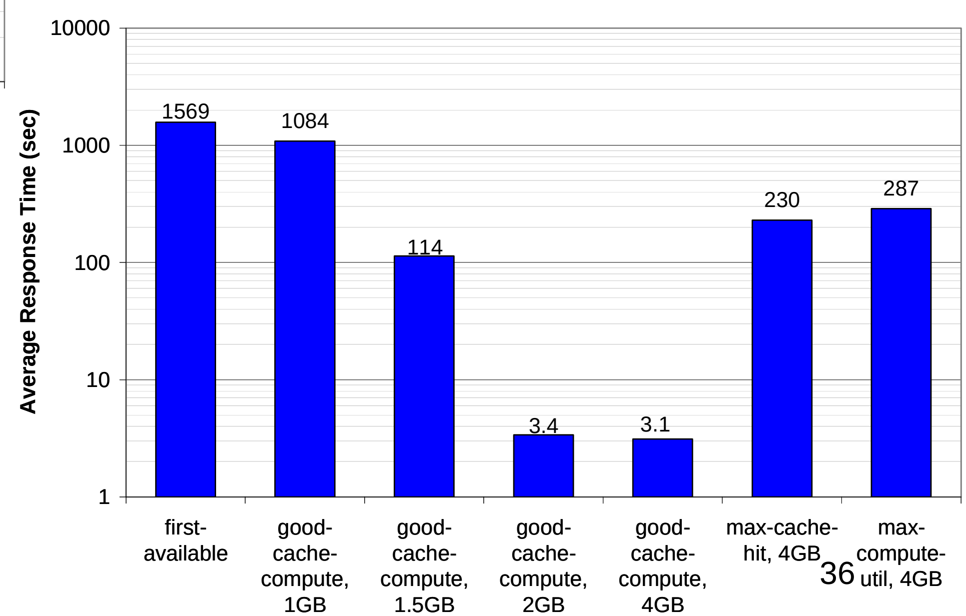



🡐 Throughput:

- Average: 14Gb/s vs 4Gb/s
- Peak: 100Gb/s vs. 6Gb/s

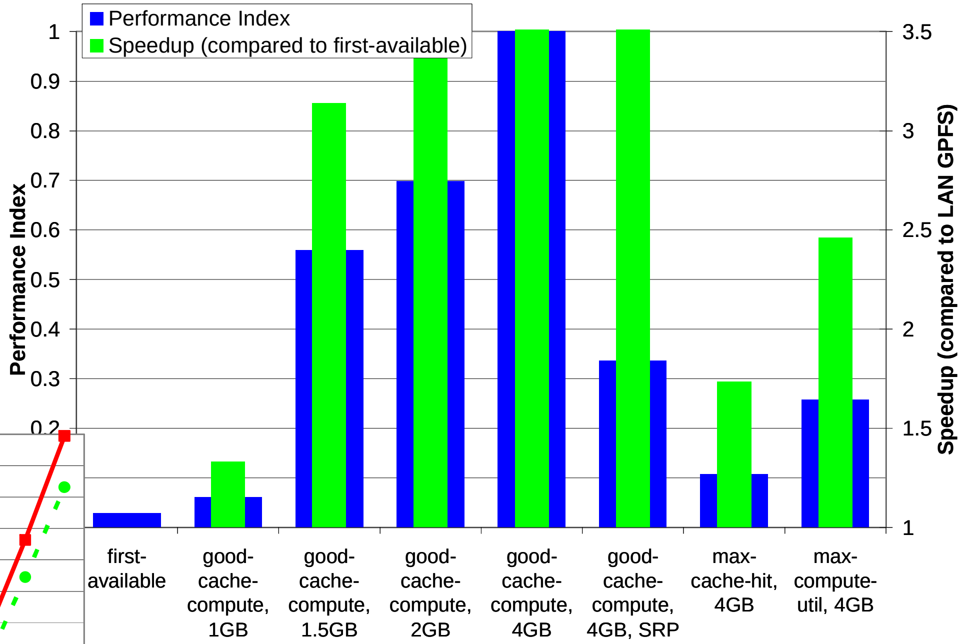
Response Time ➔

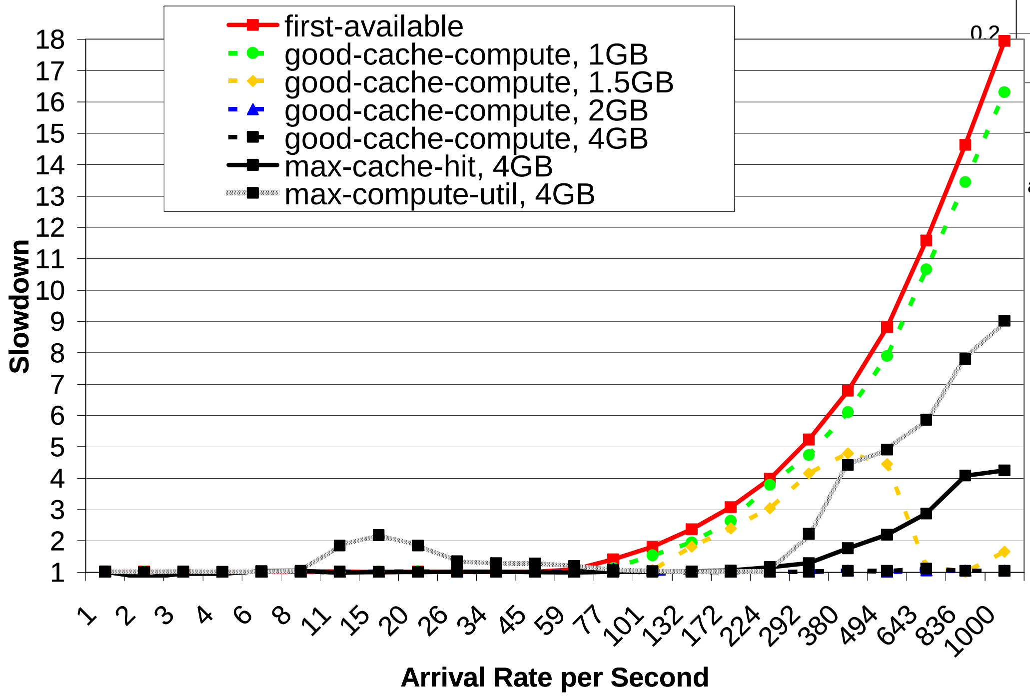
- 3 sec vs 1569 sec ➔ 506X



# Data Diffusion: Performance Index, Slowdown, and Speedup

- Performance Index:
  - 34X higher
- Speedup
  - 3.5X faster than GPFS



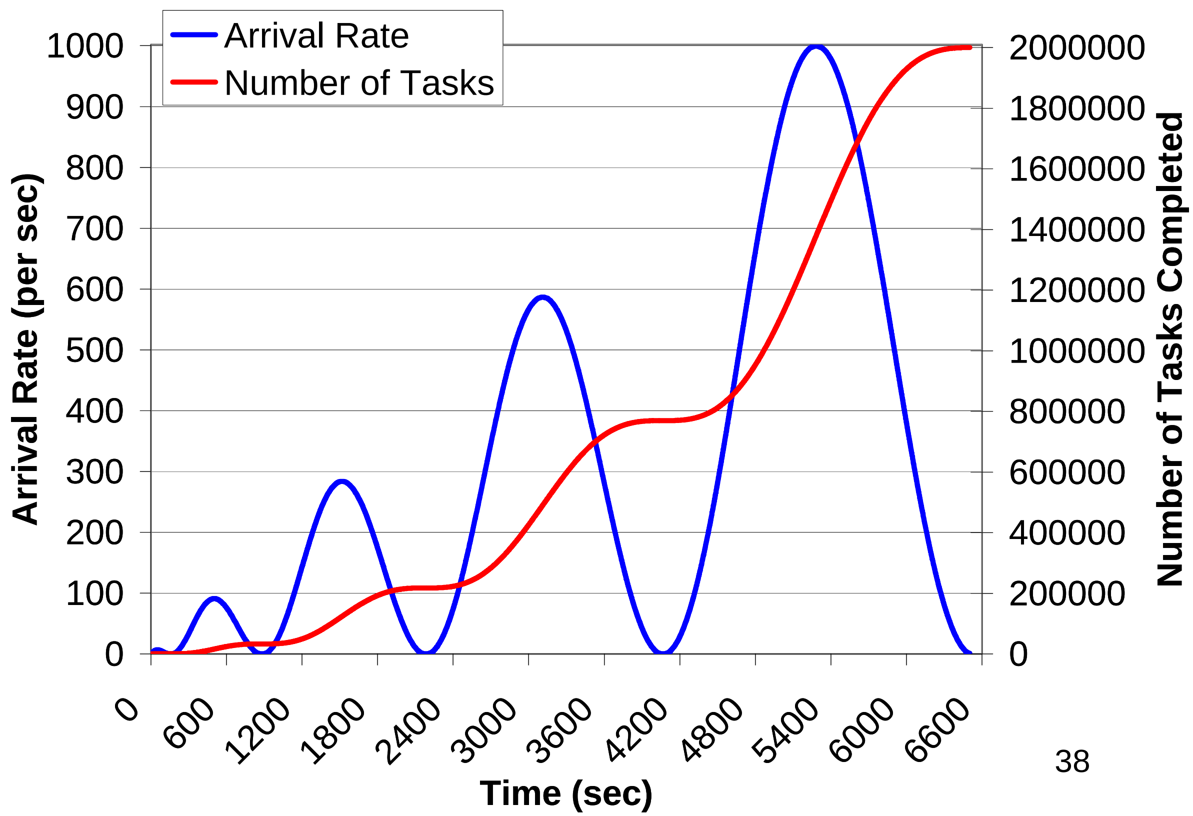




- Slowdown:
  - 18X slowdown for GPFS
  - Near ideal 1X slowdown for large enough caches

# Sin-Wave Workload

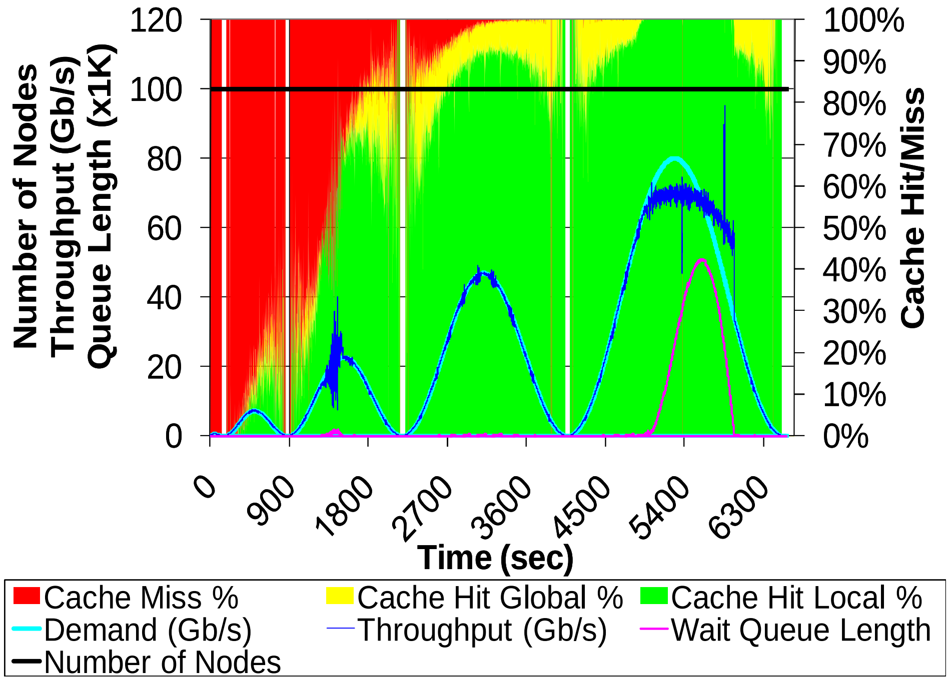
- 2M tasks
  - 10MB reads
  - 10ms compute
- Vary arrival rate:
  - Min: 1 task/sec
  - Arrival rate function:
  - Max: 1000 tasks/sec
- 200 processors
- Ideal case:
  - 6505 sec
  - 80Gb/s peak throughput

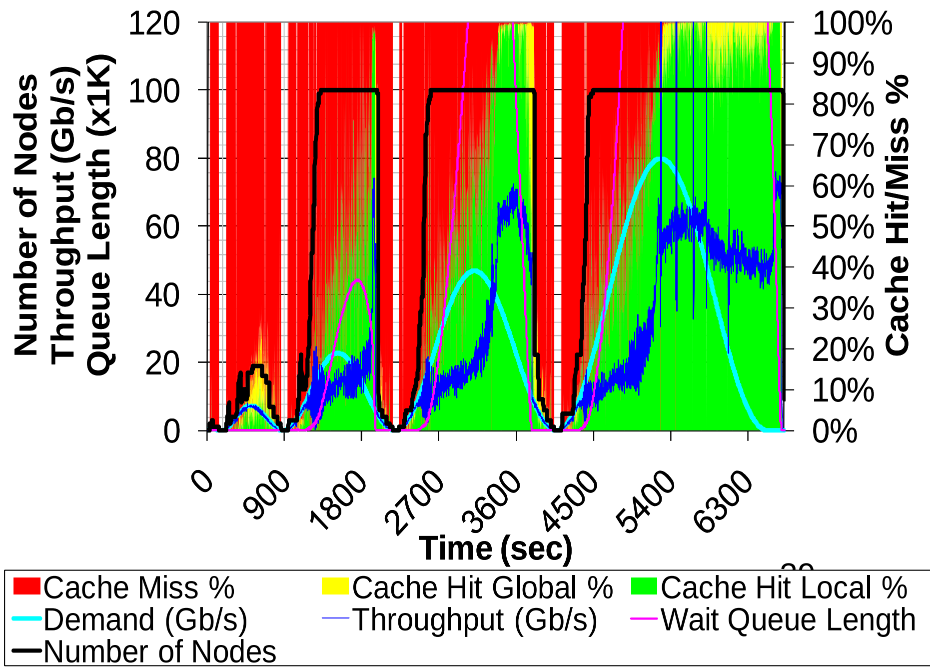
$$A = \left\lfloor (\sin(sqrt(time+0.11)*2.859678)+1)*(time+0.11)*5.705 \right\rfloor$$



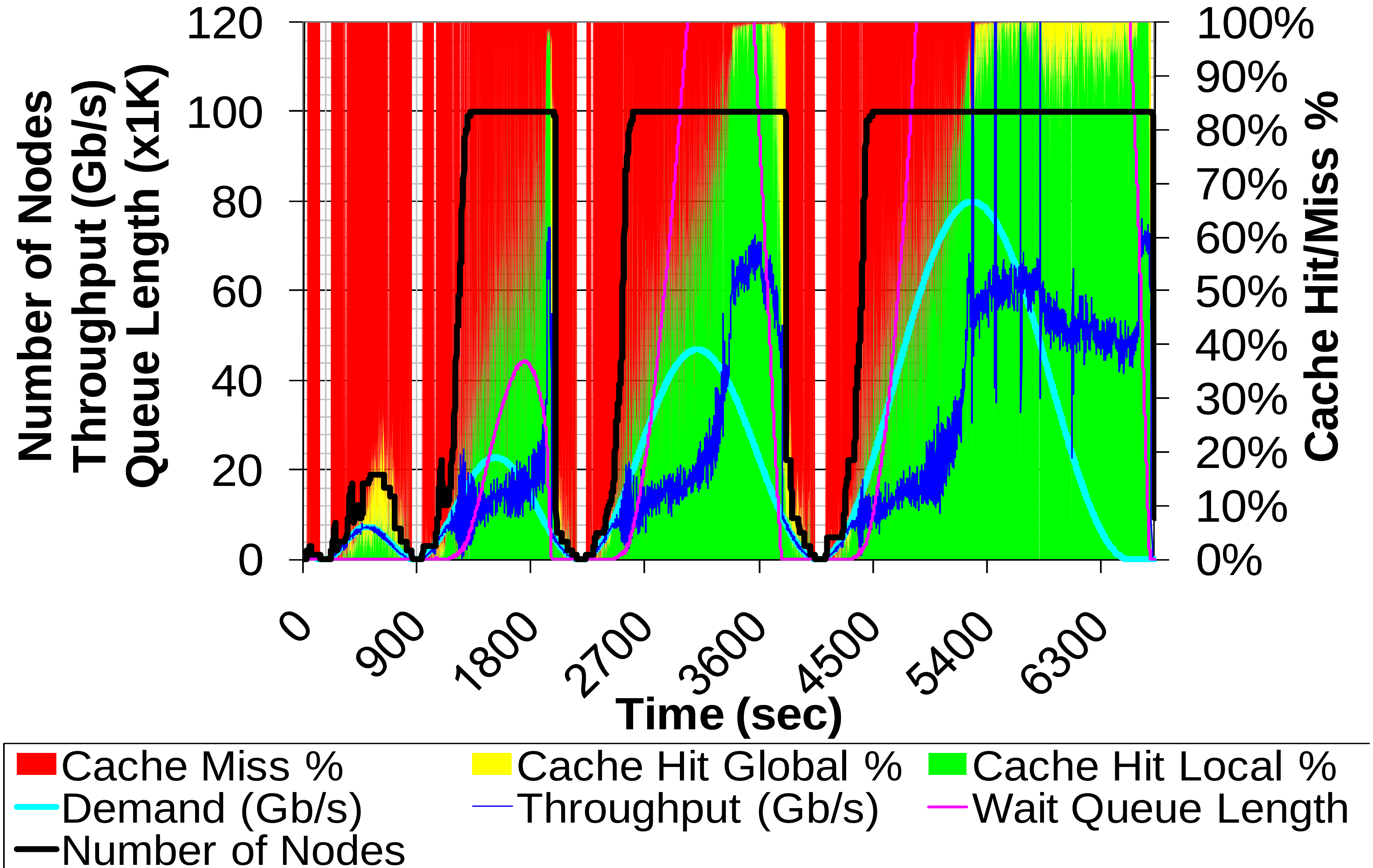
# Sin-Wave Workload

- GPFS ➔ 5.7 hrs, ~8Gb/s, 1138 CPU hrs
- DF+SRP ➔ 1.8 hrs, ~25Gb/s, 361 CPU hrs
- DF+DRP ➔ 1.86 hrs, ~24Gb/s, 253 CPU hrs



# Sin-Wave Workload



# All-Pairs Workload

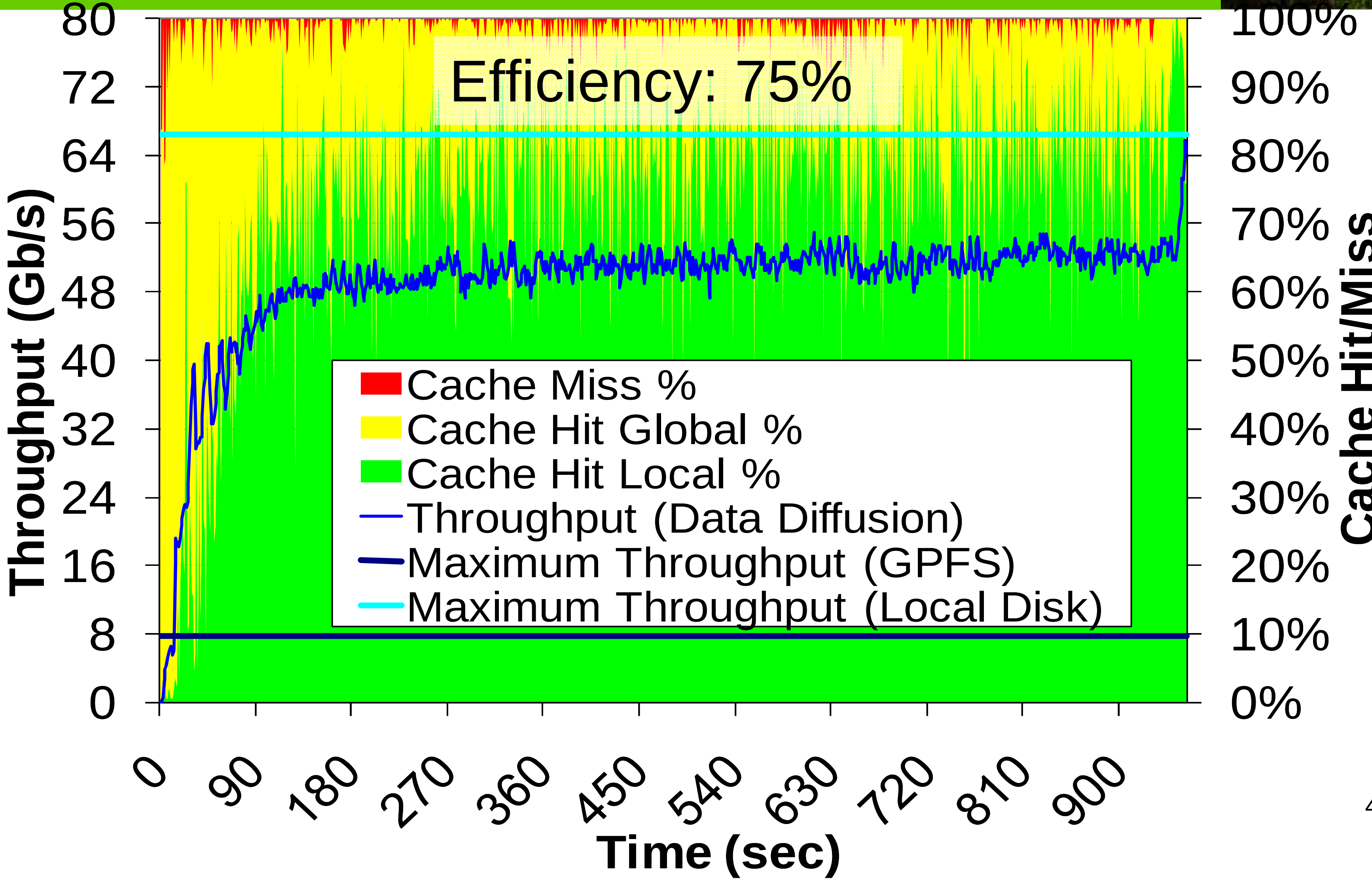
- 500x500
  - 250K tasks
  - 24MB reads
  - 100ms compute
  - 200 CPUs
- 1000x1000
  - 1M tasks
  - 24MB reads
  - 4sec compute
  - 4096 CPUs
- Ideal case:
  - 6505 sec
  - 80Gb/s peak throughput

- All-Pairs( set A, set B, function F ) returns matrix M:
- Compare all elements of set A to all elements of set B via function F, yielding matrix M, such that

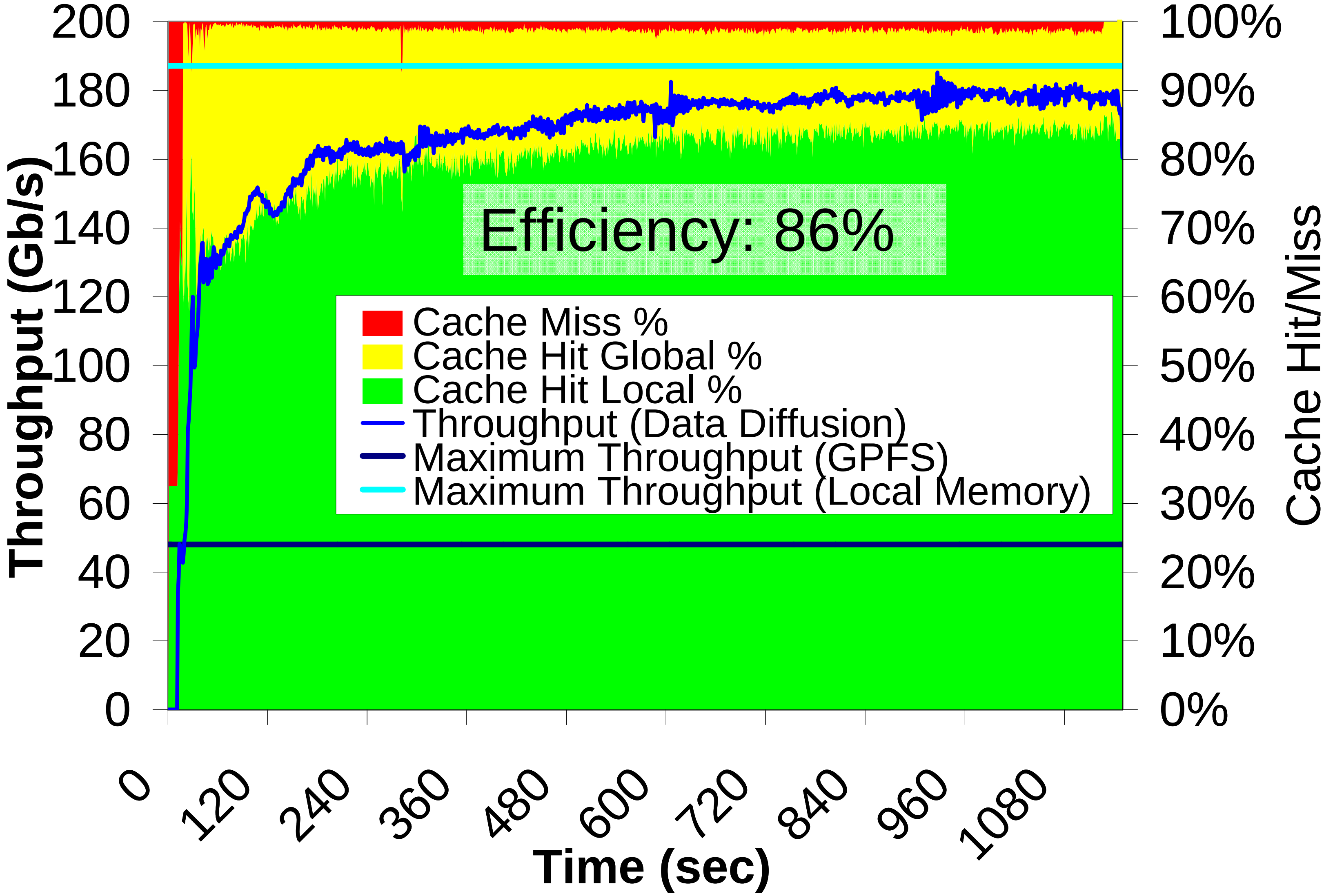
$$M[i,j] = F(A[i],B[j])$$

```
foreach $i in A
    foreach $j in B
        submit_job F $i $j
    end
end
```

# All-Pairs Workload
# 500x500 on 200 CPUs

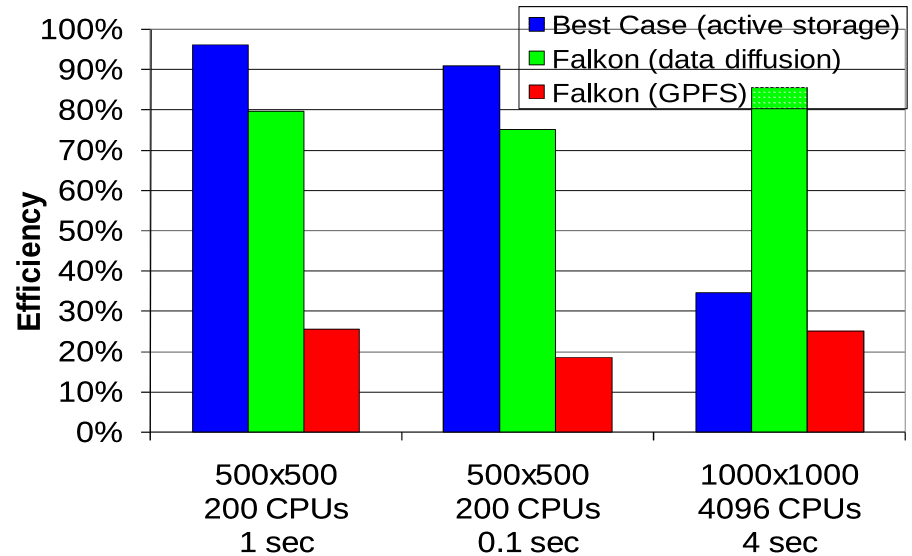


# All-Pairs Workload
# 1000x1000 on 4K emulated CPUs



# All-Pairs Workload
# Data Diffusion vs. Active Storage

- Push vs. Pull
  - Active Storage:
    - Pushes ***workload*** working set to all nodes
    - Static spanning tree
  - Data Diffusion
    - Pulls ***task*** working set
    - Incremental spanning forest




| Experiment | Approach | Local Disk/Memory (GB) | Network (node-to-node) (GB) | Shared File System (GB) |
|---|---|---|---|---|
| 500x500 200 CPUs 1 sec | Best Case (active storage) | 6000 | 1536 | 12 |
| | Falkon (data diffusion) | 6000 | 1698 | 34 |
| 500x500 200 CPUs 0.1 sec | Best Case (active storage) | 6000 | 1536 | 12 |
| | Falkon (data diffusion) | 6000 | 1528 | 62 |
| 1000x1000 4096 CPUs 4 sec | Best Case (active storage) | 24000 | 12288 | 24 |
| | Falkon (data diffusion) | 24000 | 4676 | 384 |

# All-Pairs Workload
# Data Diffusion vs. Active Storage

- Best to use active storage if
  - Slow data source
  - Workload working set fits on local node storage
  - Good aggregate network bandwidth
- Best to use data diffusion if
  - Medium to fast data source
  - Task working set << workload working set
  - Task working set fits on local node storage
  - Good aggregate network bandwidth
- If task working set does not fit on local node storage
  - Use parallel file system (i.e. GPFS, Lustre, PVFS, etc)

# Limitations of Data Diffusion

- Needs Java 1.4+
- Needs IP connectivity between hosts
- Needs local storage (disk, memory, etc)
- Per task workings set must fit in local storage
- Task definition must include input/output files metadata
- Data access patterns: write once, read many

# Related Work: Data Management

- [*Beynon01*]: **DataCutter**
- [*Ranganathan03*]: **Simulations**
- [*Ghemawat03,Dean04,Chang06*]: **BigTable**, **GFS**, **MapReduce**
- [*Liu04*]: **GridDB**
- [*Chervenak04,Chervenak06*]: **RLS** (Replica Location Service), **DRS** (Data Replication Service)
- [*Tatebe04,Xiaohui05*]: **GFarm**
- [*Branco04,Adams06*]: **DIAL/ATLAS**
- [*Kosar06*]: **Stork**
- [*Thain08*]: **Chirp/Parrot**

***Conclusion:*** None focused on the co-location of storage and generic black box computations with data-aware scheduling while operating in a dynamic environment

# Scaling from 1K to 100K CPUs without Data Diffusion

- At 1K CPUs:
  - 1 Server to manage all 1K CPUs
  - Use shared file system extensively
    - Invoke application from shared file system
    - Read/write data from/to shared file system
- At 100K CPUs:
  - N Servers to manage 100K CPUs (1:256 ratio)
  - Don’t trust the application I/O access patterns to behave optimally
    - Copy applications and input data to RAM
    - Read input data from RAM, compute, and write results to RAM
    - Archive all results in a single file in RAM
    - Copy 1 result file from RAM back to GPFS
- Great potential for improvements
  - Could leverage the Torus network for high aggregate bandwidth
  - Collective I/O (CIO) Primitives
  - Roadblocks: machine global IP connectivity, Java support, and time

# Mythbusting

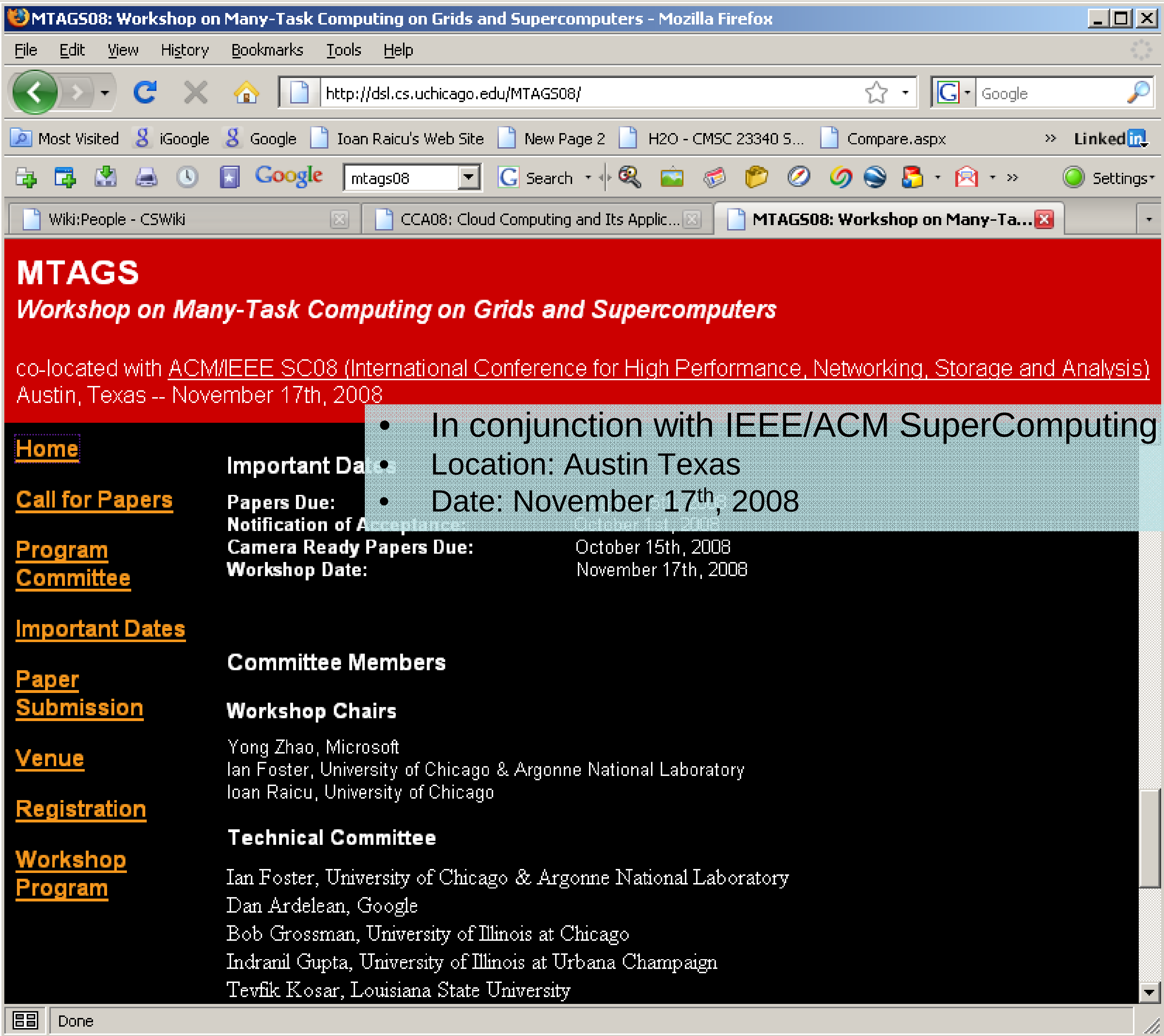
- ~~Embarrassingly~~ Happily parallel apps are trivial to run
  - Logistical problems can be tremendous
- Loosely coupled apps do not require “supercomputers”
  - Total computational requirements can be enormous
  - Individual tasks may be tightly coupled
  - Workloads frequently involve large amounts of I/O
  - Make use of idle resources from “supercomputers” via backfilling
  - Costs to run “supercomputers” per FLOP is among the best
    - BG/P: 0.35 gigaflops/watt **(higher is better)**
    - SiCortex: 0.32 gigaflops/watt
    - BG/L: 0.23 gigaflops/watt
    - x86-based HPC systems: an order of magnitude lower
- Loosely coupled apps do not require specialized system software
- Shared file systems are good for all applications
  - They don’t scale proportionally with the compute resources
  - Data intensive applications don’t perform and scale well

# Conclusions & Contributions

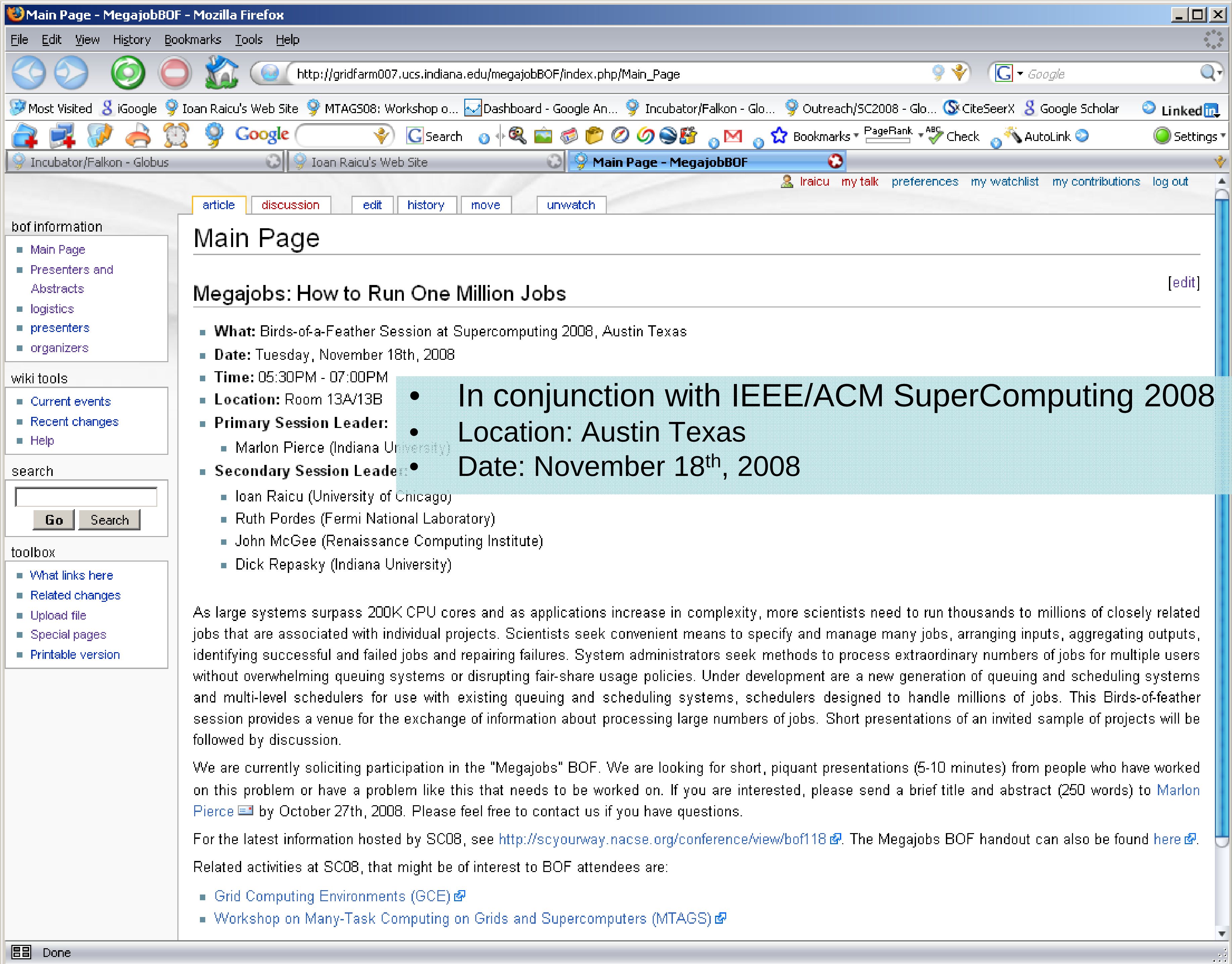
- Defined an *abstract model for performance efficiency of data analysis workloads* using data-centric task farms
- Provide a reference implementation (Falkon)
  - Use a streamlined dispatcher to increase task throughput by several orders of magnitude over traditional LRMs
  - Use multi-level scheduling to reduce perceived wait queue time for tasks to execute on remote resources
  - Address data diffusion through co-scheduling of storage and computational resources to improve performance and scalability
  - Provide the benefits of dedicated hardware without the associated high cost
  - Show effectiveness of data diffusion:
    - real large-scale astronomy application and a variety of synthetic workloads

# More Information

- More information: http://people.cs.uchicago.edu/~iraicu/
- Related Projects:
  - Falkon:
    - http://dev.globus.org/wiki/Incubator/Falkon
  - AstroPortal:
    - http://people.cs.uchicago.edu/~iraicu/projects/Falkon/astro_portal.htm
  - Swift:
    - http://www.ci.uchicago.edu/swift/index.php
- Funding:
  - NASA: Ames Research Center, Graduate Student Research Program (GSRP)
  - DOE: Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy
  - NSF: TeraGrid




- In conjunction with IEEE/ACM SuperComputing 2008
- Location: Austin Texas
- Date: November 17th, 2008




- In conjunction with IEEE/ACM SuperComputing 2008
- Location: Austin Texas
- Date: November 18th, 2008