Data Management in Parallel Scripting

Zhao Zhang
11/11/2012

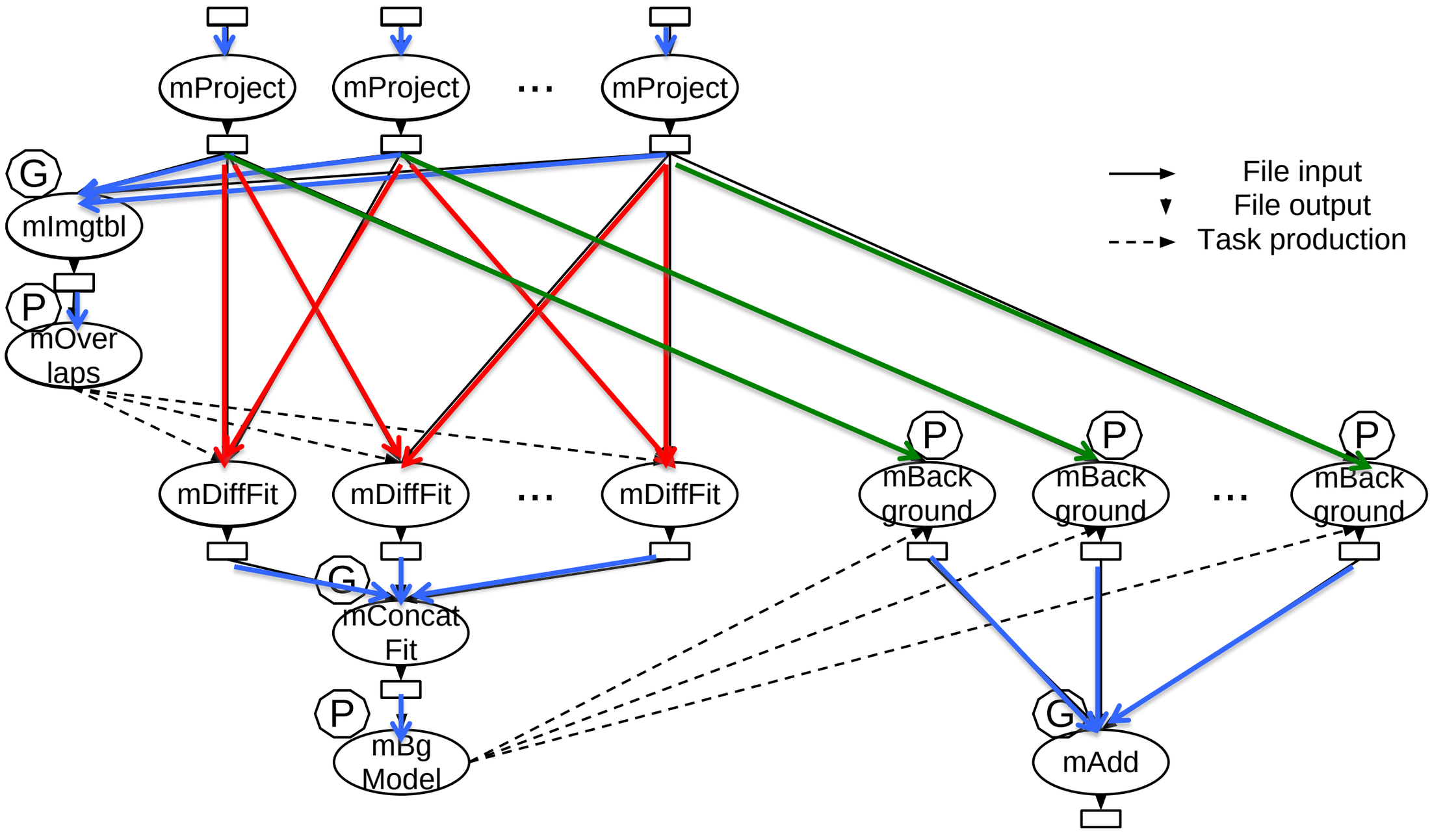
Problem Statement

- Definition:

  MTC applications are those applications in which existing sequential or parallel programs are linked by files output by one program being used as input by others.

Problem Statement

- Application Example: Montage



Problem Statement

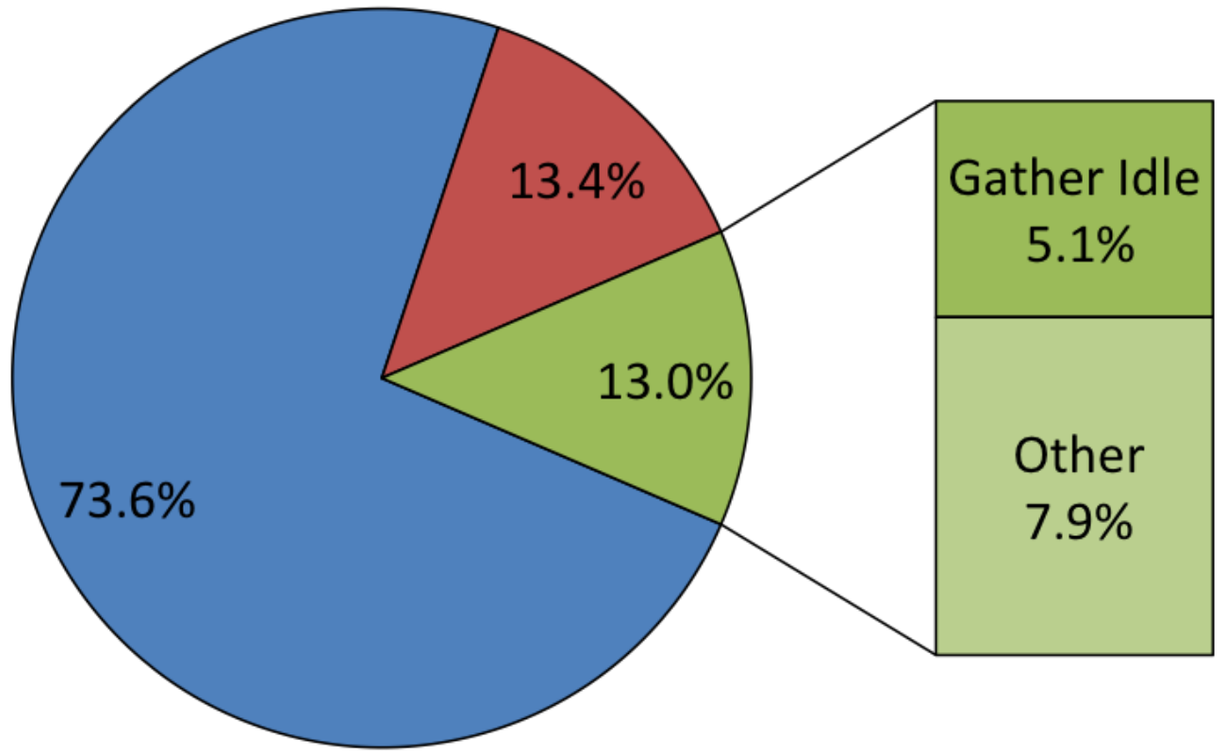
- How to run the Montage computation on large scale computers?
  - Solution 1: Rewrite the application as monolithic program using a parallel library or language such as MPI or PGAS. Communications that originally occurred via filesystem operations then occur via messaging.
  - Pros:
    - MPI is widely deployed
  - Cons:
    - Labor intensive
    - Code maintenance
    - Fault intolerance

Problem Statement

- How to run the Montage computation on large scale computers?
  - Solution 2: build the program with parallel scripting, e.g. Swift/T (http://sites.google.com/site/exmcomputing/).
    - The user represents the task graph by a script, which is compiled to tasks when running. Files are used for intertask communication, and are stored on shared filesystem for further access.
  - Pros:
    - Doesn't modify the original application
    - Updating the application version is seamless unless the program's interface changes substantially.
    - Can handle faults
  - Cons:
    - Relatively high latency when accessing files compared to in-memory data
    - The volume and frequency of filesystem operations generated by a large scripting computation are often overwhelming.

Problem Statement

- Is "Mapping computation to computing resources" a sufficiently efficient way to enable MTC application on large scale computers?



Execution time distribution of Montage on 512 BG/P CPU cores

Problem Statement

- Problems:
  - Persistent file system I/O has a large amount of metadata traffic and I/O traffic.
  - There is unnecessary intermediate I/O traffic.
  - Some of the idle time is due to data flow patterns.

Approaches

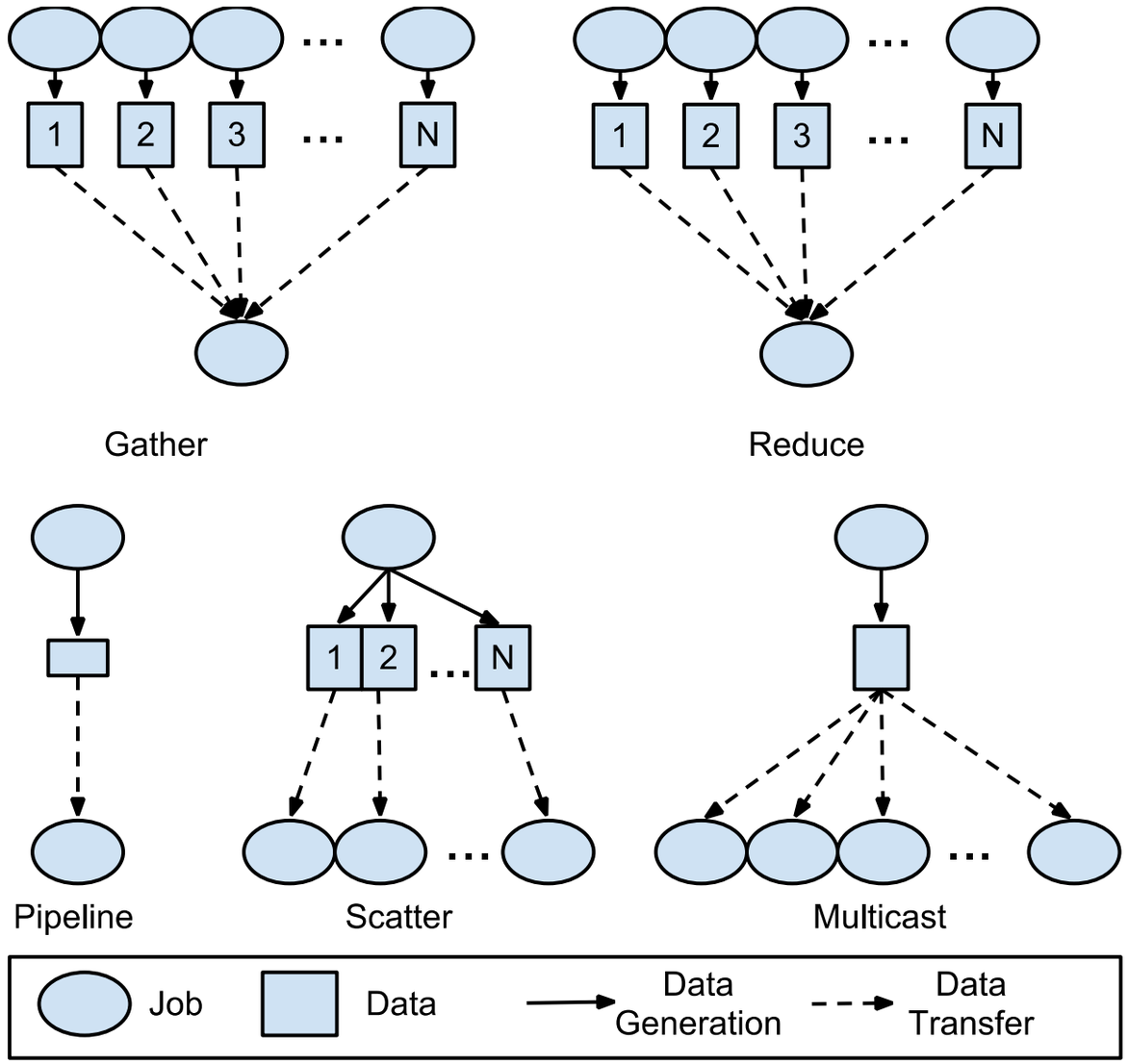
- To understand the I/O behavior of MTC applications, we can study across many scientific applications.
- To speed up the persistent file system I/O, we can use a collective scheme that maximize the I/O performance.
- To eliminate the unnecessary I/O traffic, we can use an intermediate file system by aggregating the local storage of the compute nodes, and cache intermediate data on it.
- To speed up the intermediate data movement, we can build a group of data transfer schemes to accommodate data flow patterns

Publications

- Understanding the MTC application I/O
  - "Many-task computing and blue waters", DS Katz, TG Armstrong, Z Zhang, M Wilde, JM Wozniak - arXiv preprint arXiv:1202.3943, 2012
- Persistent file system Collective I/O
  - "Design and evaluation of a collective IO model for loosely coupled petascale programming" Z Zhang, A Espinosa, K Iskra, I Raicu, I Foster, M Wilde - MTAGS 2008
- Intermediate data caching
  - "A Workflow-Aware Storage System: An Opportunity Study", E Vairavanathan, S Al-Kiswany, L Costa, Z Zhang, DS Katz, M Wilde, M Ripeanu - CCGrid 2012
  - "Design and Analysis of Data Management in Scalable Parallel Scripting", Z Zhang, DS Katz, JM Wozniak, A Espinosa, I Foster – SC 2012
- Speedup data movement in data flow patterns
  - "A Workflow-Aware Storage System: An Opportunity Study", E Vairavanathan, S Al-Kiswany, L Costa, Z Zhang, DS Katz, M Wilde, M Ripeanu - CCGrid 2012
  - "Design and Analysis of Data Management in Scalable Parallel Scripting", Z Zhang, DS Katz, JM Wozniak, A Espinosa, I Foster - SC 2012

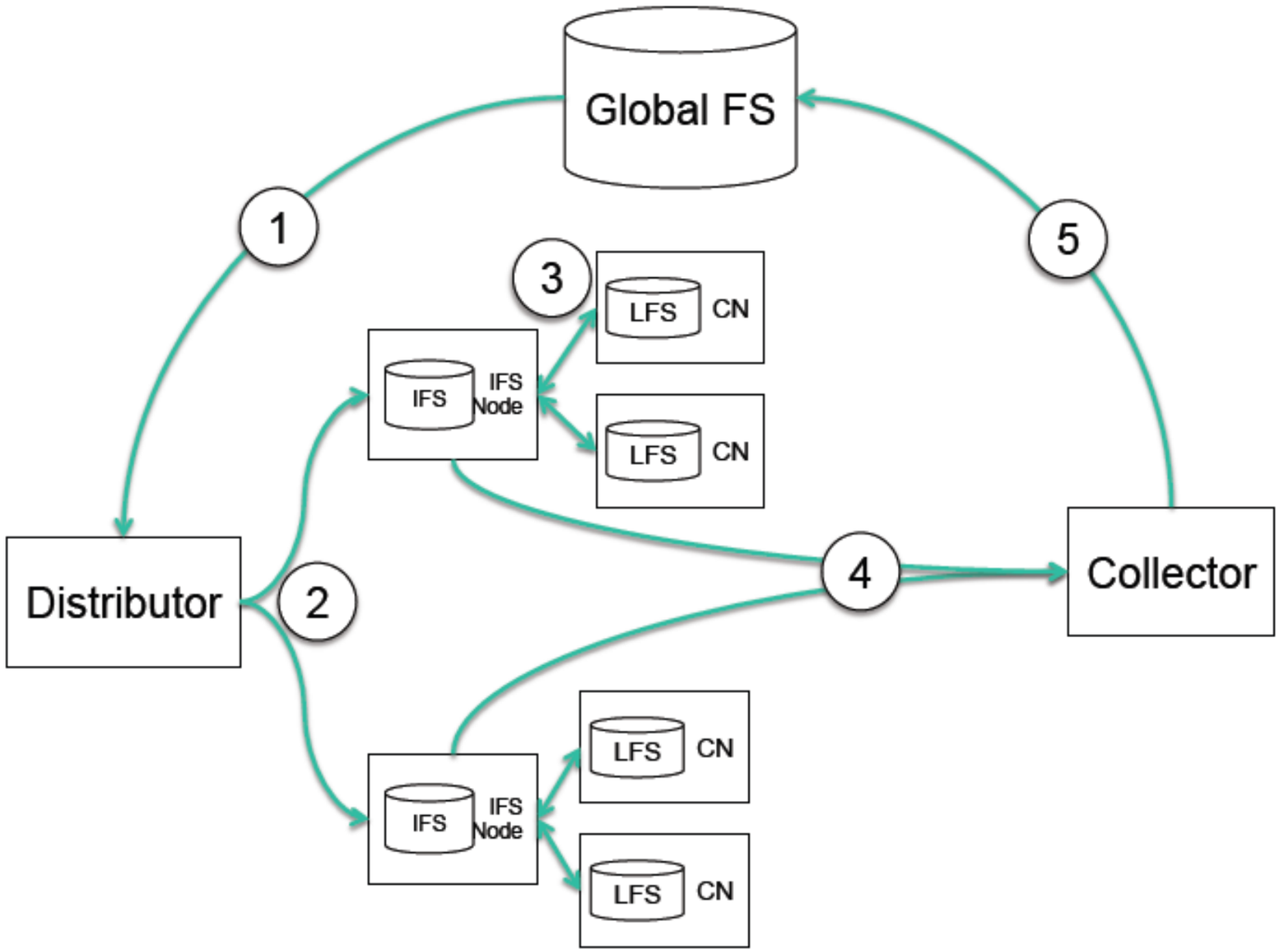
Dataflow patterns

- Understanding the MTC application I/O
  - Data flow patterns
  - A quantitative study – in progress



Design

- Persistent file system Collective I/O -- MTAGS '08



Options

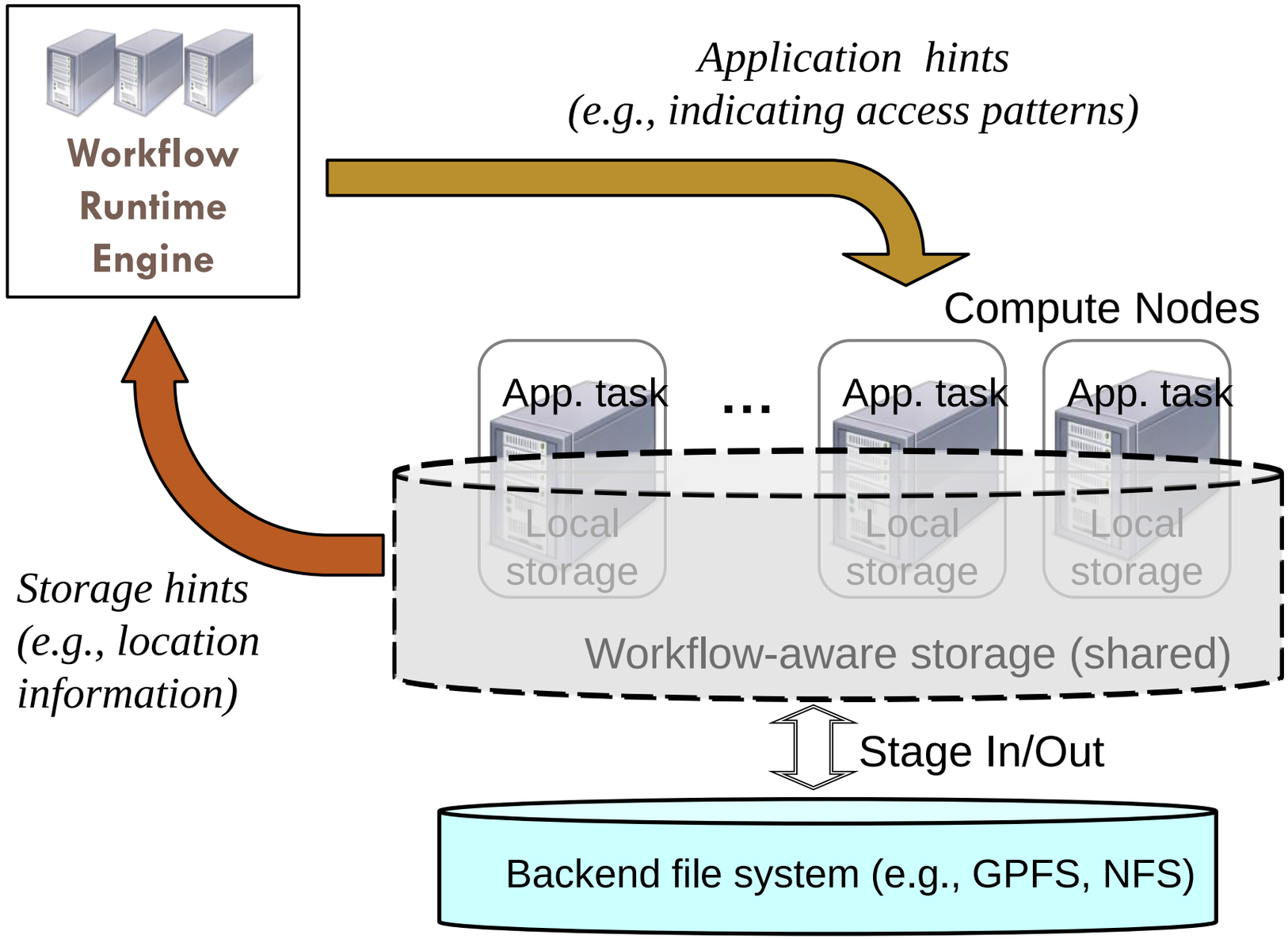
- Intermediate data caching
- Questions to answer:
  - Data-aware scheduling enabled or not?
  - POSIX compatible or not?
  - Stripe based or file based?
  - Dataflow pattern optimized transfer or one size fits all?
  - Dataflow pattern identification by users, compiler, or system?

Design -- MosaStore

- The MosaStore approach
  - Data-aware scheduling enabled – by providing the engine the file location
  - POSIX compatible – file location encoded in POSIX extra file descriptor
  - Stripe based
  - Dataflow pattern specific optimization – block placement schemes for various patterns
  - User pass in the hints in the parallel scripting language.

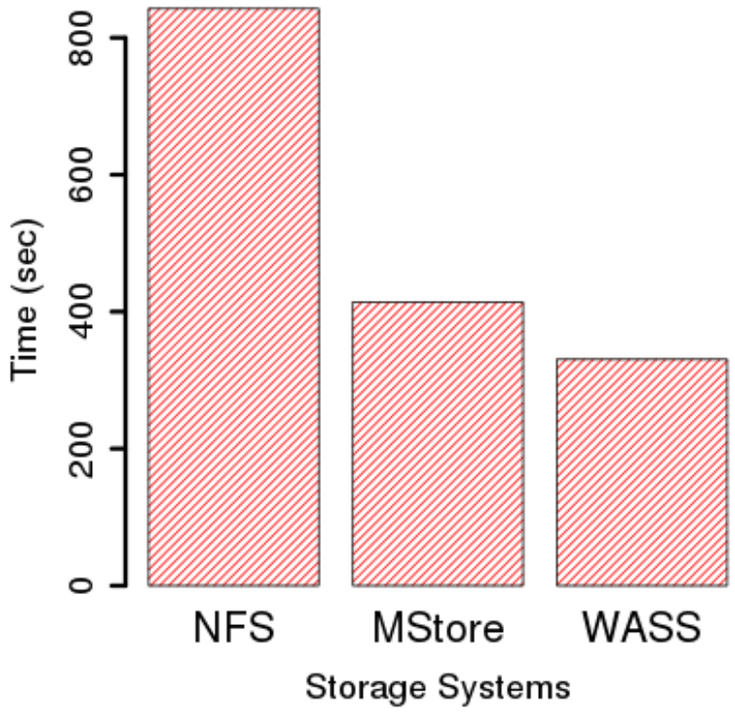
Design -- MosaStore

- The MosaStore approach

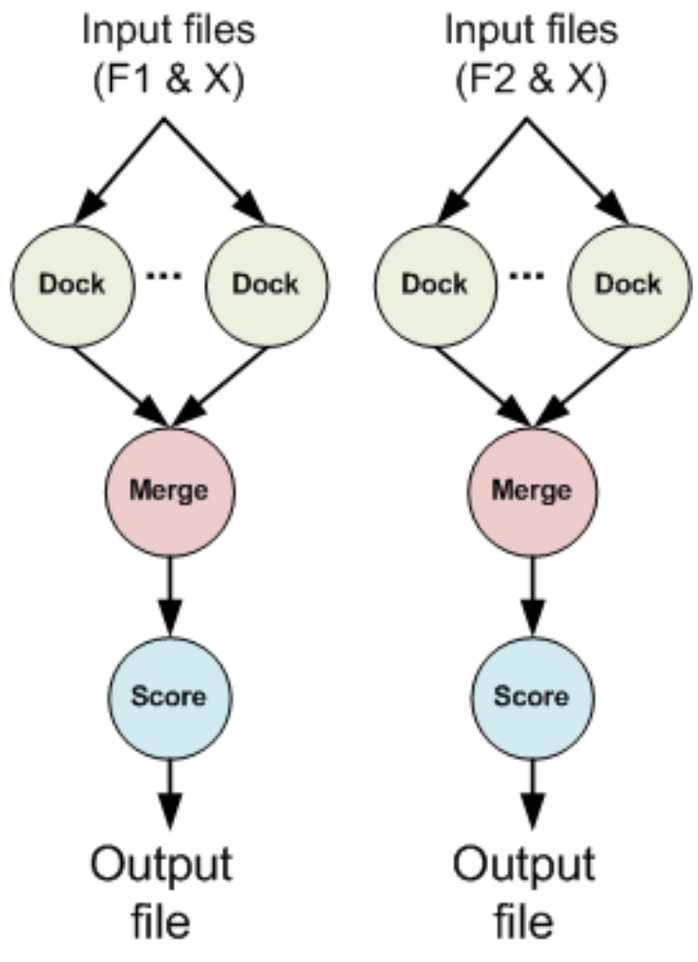


Results -- MosaStore

- The MosaStore approach
  - Tests run on a cluster of 20 nodes, with NFS as persistent file system.



Total application time on three different systems



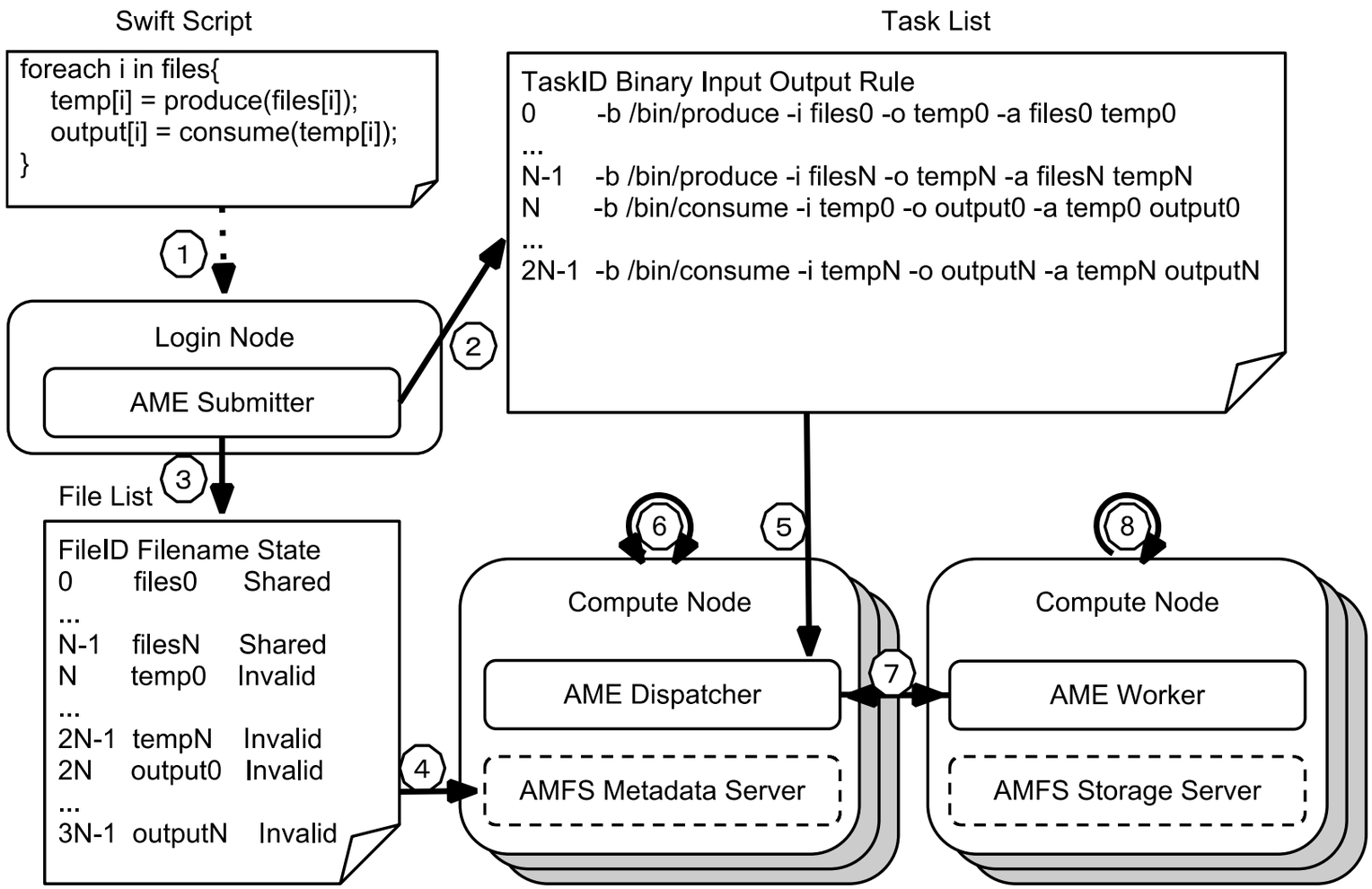
ModFTDock workflow

Design -- AME/AMFS

- The AME/AMFS approach
  - Data-aware scheduling enabled – by providing the engine the file location
  - POSIX imcompatible – file location encoded in TCP message
  - File based
  - Dataflow pattern specific optimization – tree topology based data movement at large scale
  - Runtime system detect the data flow pattern based on task information

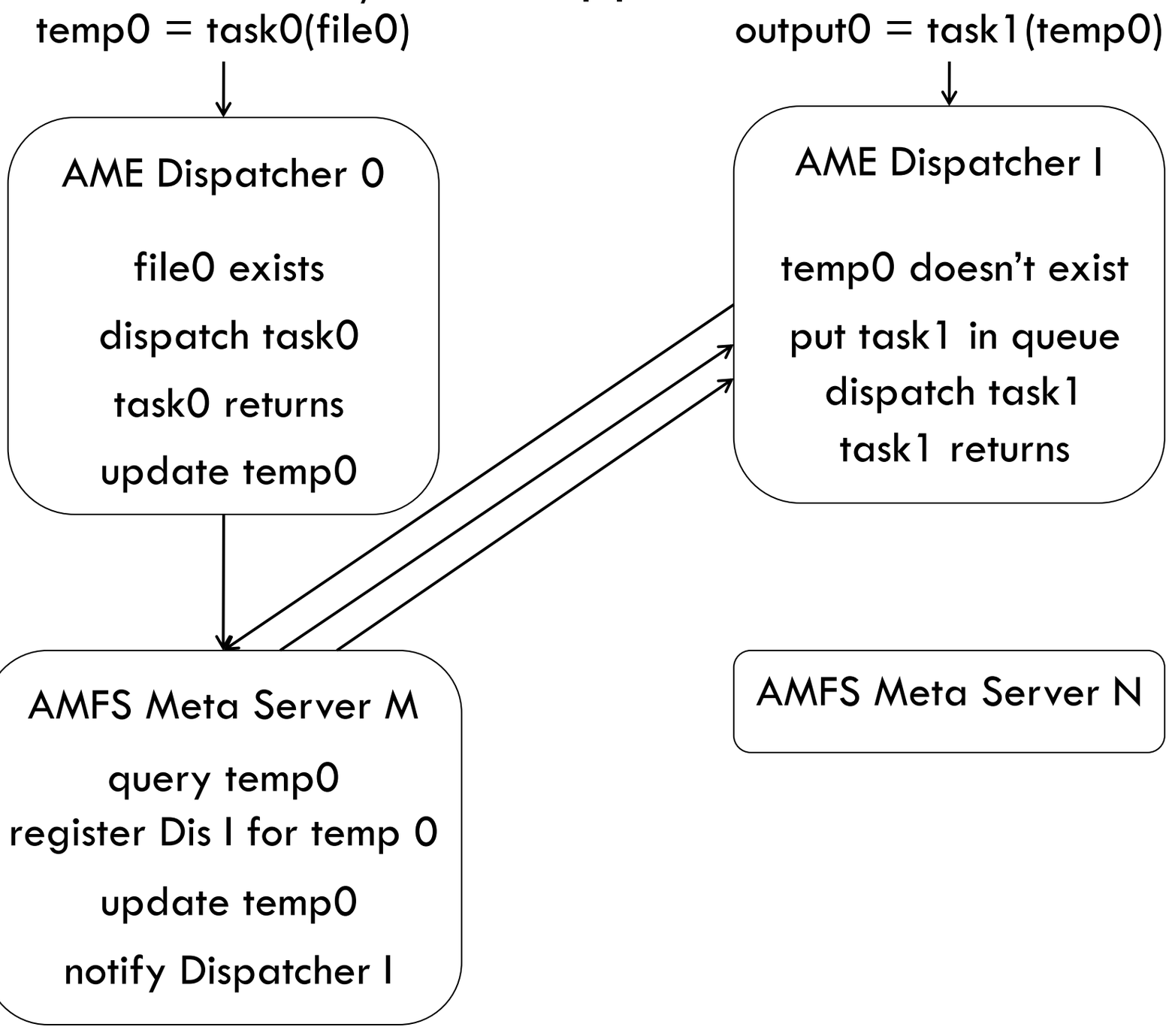
Design -- AME/AMFS

- The AME/AMFS approach



Design -- AME/AMFS

- The AME/AMFS approach

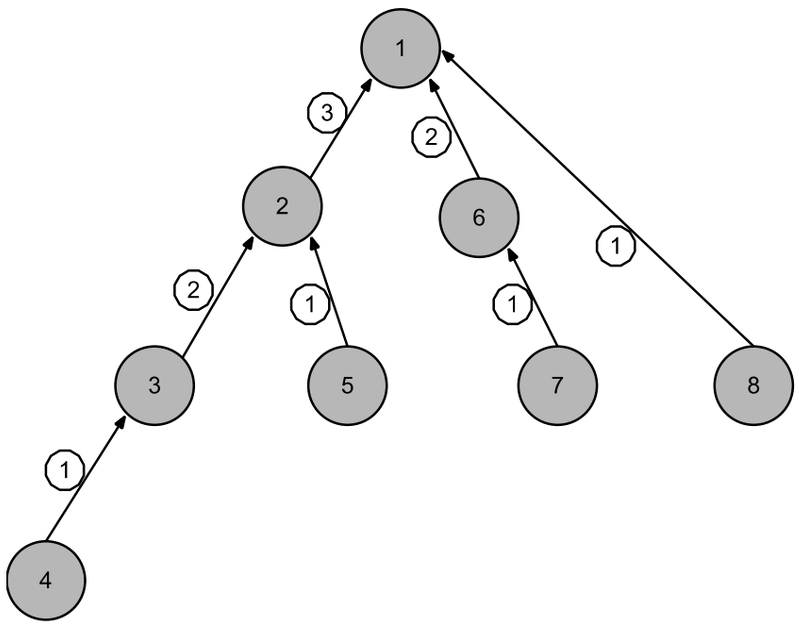


Files are mapped to the Meta Servers by a hash function.

hashvalue = hash(filename)
ServerID = hashvalue % count

Design -- AME/AMFS

- The AME/AMFS approach
  - The dispatcher has the information to detect the data flow pattern at runtime.
  - Speedup data movement at large scale (e.g. Sync Gather and Async Gather)



N: Number of nodes

S: Total bytes transferred

M: Number of files

a: latency overhead

b: bandwidth overhead per byte

c: overhead per file

$$SyncGather: T = (\log_2 N) * a + \frac{N-1}{N} * S * b + (M-1) * c$$

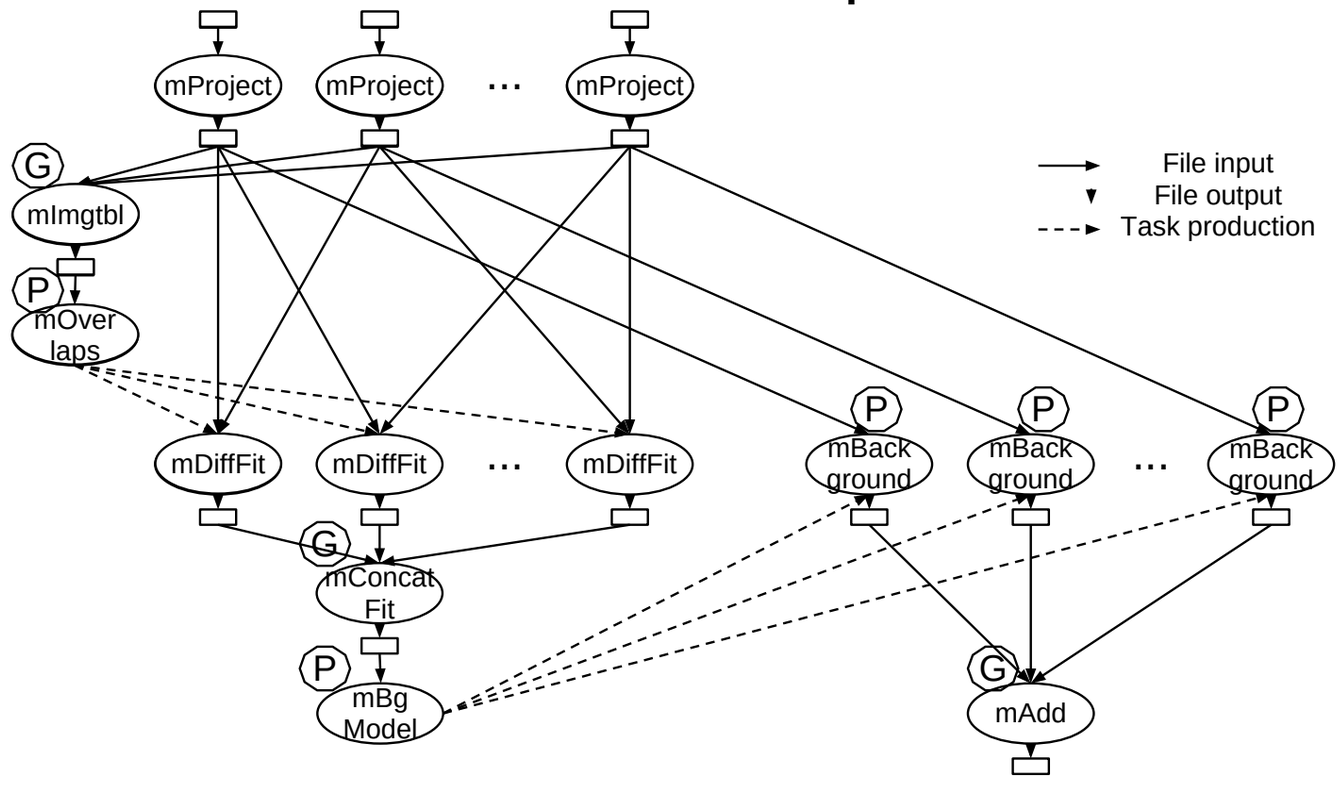
$$AsyncGather: T = (N-1) * a + \frac{N-1}{N} * S * b + (M-1) * c$$

Design -- AME/AMFS

- The AME/AMFS approach
  - But, how to detect the patterns when the information is distributed across dispatchers?
  - Please come to my talk: “Design and Analysis of Data Management in Scalable Parallel Scripting”, Thursday, Nov 15th, 1:30PM - 2:00PM, 255-EF

Results -- AME/AMFS

- Montage
  - Base Case: Staging, runs in 45% of the time as the MPI implementation
  - Four techniques tested:
    - Data cache
    - Data aware scheduling
    - Collective gather
    - Asynchronous gather
  - Test setup
    - a 6x6 degree mosaic of the 2MASS data set with Galaxy m101 as the center
    - on 512 compute nodes

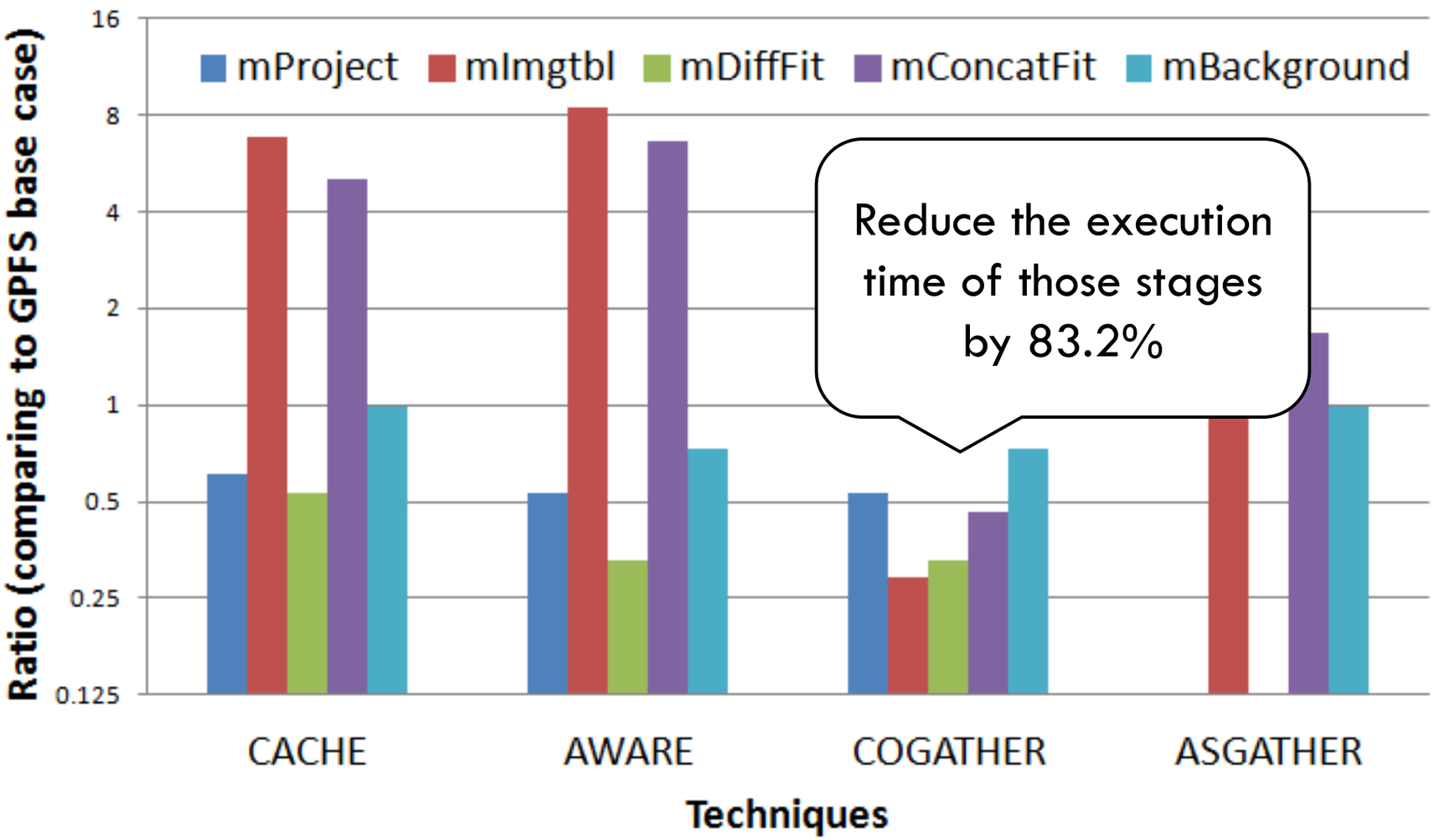


MONTAGE STAGE TASKS, INPUTS, OUTPUTS, INPUT AND OUTPUT SIZE

Stage	# Tasks	# In	# Out	In (MB)	Out (MB)
mProject	1319	1319	2638	2800	5500
mImgtbl	1	1319	1	2800	0.81
mDiffFit	3883	7766	3883	31000	3900
mConcatFit	1	3883	1	3900	0.32
mBackground	1297	1297	1297	5200	3700

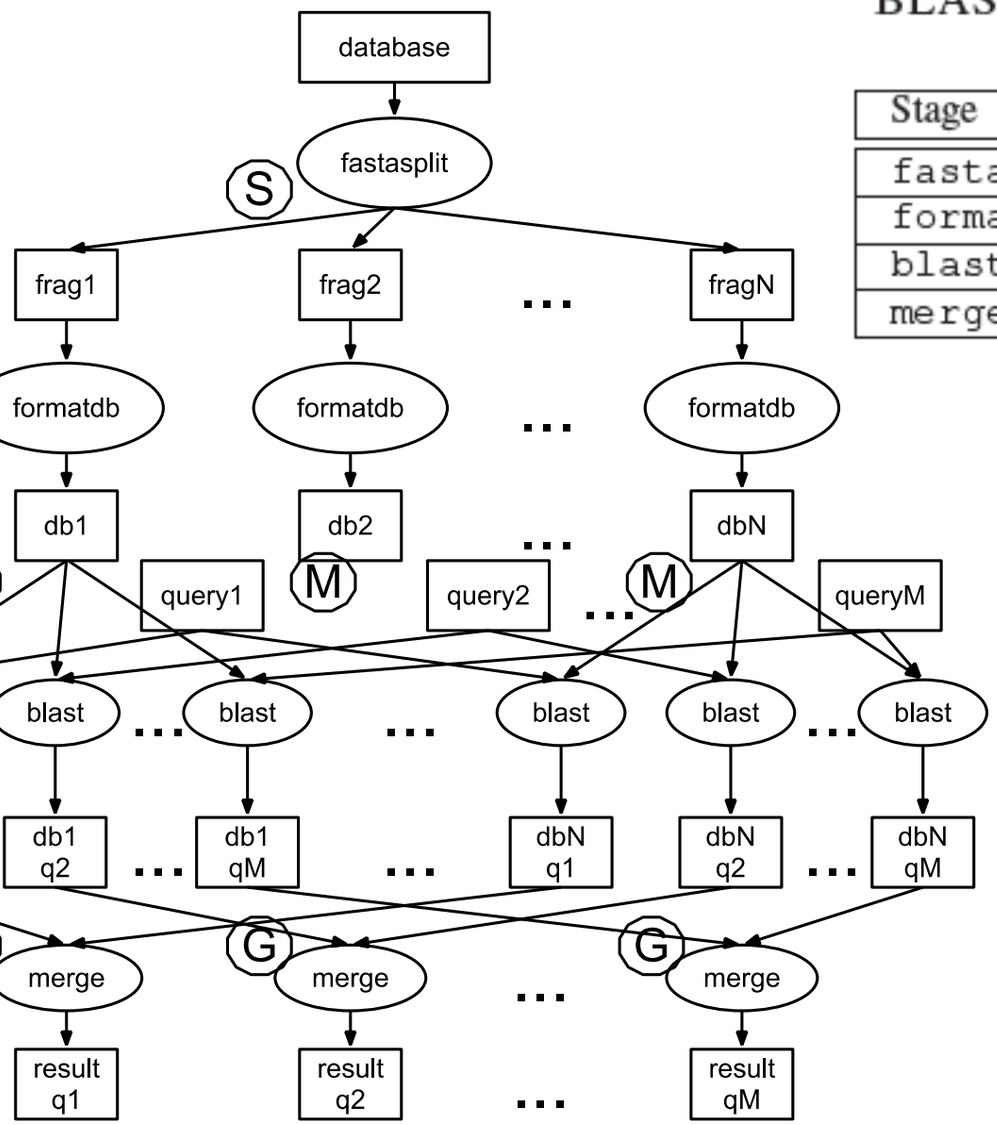
Results -- AME/AMFS

- Montage
  - Those bars that are less than 1 show improvements.
  - GPFS base case refers to Staging input/output data from/to GPFS



Results -- AME/AMFS

- BLAST

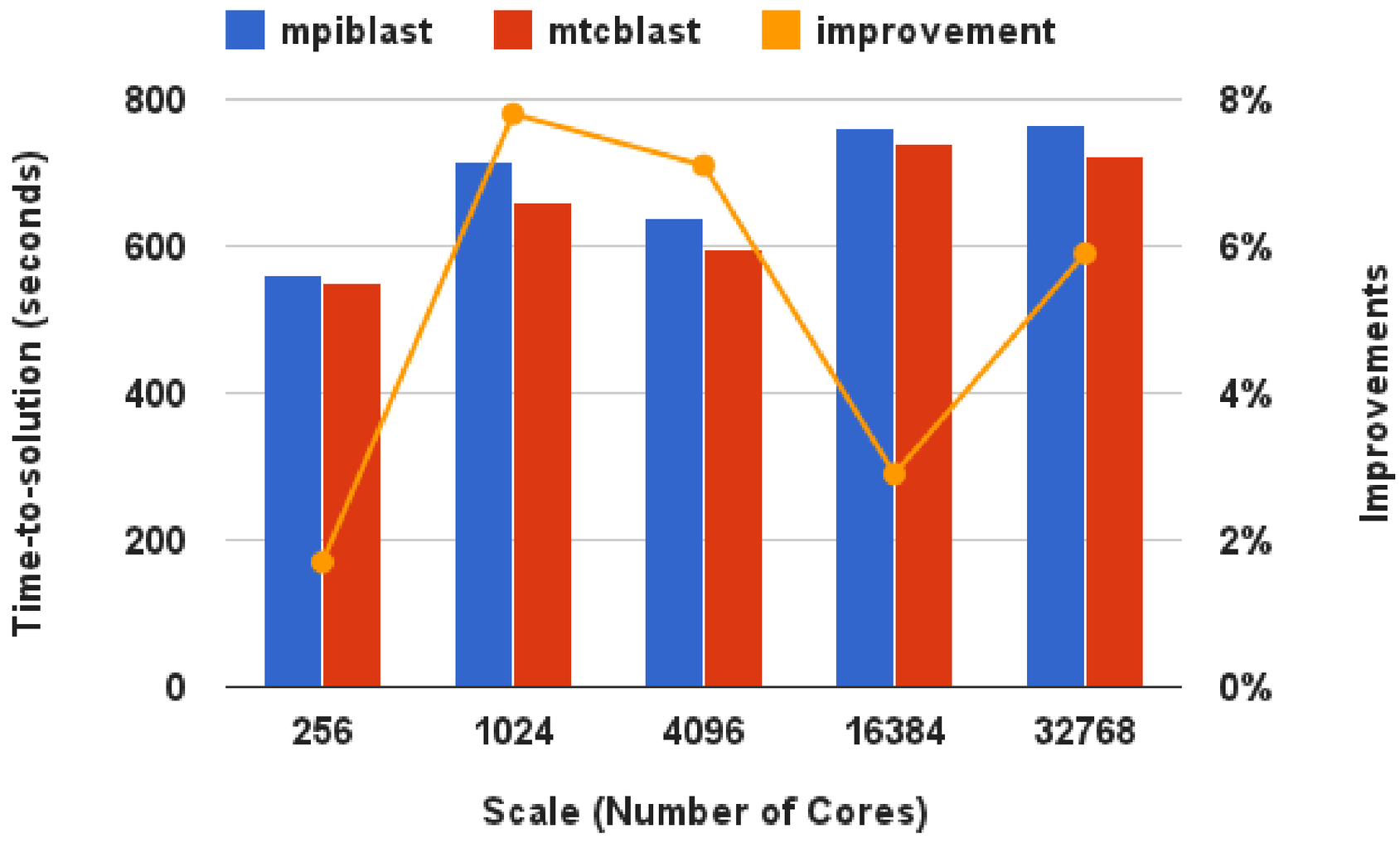


BLAST STAGE TASKS, INPUTS, OUTPUTS, AND INPUT AND OUTPUT SIZE

Stage	# Tasks	# In	# Out	In (MB)	Out (MB)
`fastasplitn`	1	1	N	4039	4039
`formatdb`	N	N	3N	4039	4400
`blastp`	N*M	N+M	N*M	73*N*M	2.4*N*M
`merge`	M	N*M	M	2.4*N*M	4.8*M

Results -- AME/AMFS

- mtcBLAST vs. mpiBLAST



Comparison of the two approaches

Design Choice	MosaStore	AMFS
Locality Support	Yes, by providing the engine the locality info	Yes, by providing the engine the locality info
POSIX Compatibility	Yes, by using the extra field of POSIX file descriptor	No, custom API
Stripe/file based	Stripe based	File Based
Data movement at large scale	Block based data placement	Tree topology parallel data movement
Dataflow pattern detection	User inputs/Compiler detection	Runtime detection

Conclusion

- The data management system of parallel scripting should consider data locality.
- Data movement optimization for dataflow patterns improves the overall application performance.
- Runtime system has sufficient information to detect dataflow patterns.
- Technical solutions should consider scalability.

Future work

- Finish the quantitative MTC applications I/O profile study.
- Evaluate the existing ideas and make design decision.
- Integrate the data management system with Swift/T (http://www.mcs.anl.gov/exm/local/guides/swift.html) as the parallel scripting language.
- Drive more applications through Swift/T with data management.

Acknowledgements

- ExM Team:
  - Michael Wilde
  - Daniel S. Katz
  - Matei Ripeanu
  - Ian Foster
  - Justin Wozniak
  - Tim Armstrong
  - Emalayan Vairavanathan
  - Samer Al-Kiswany
- Parallel BLAST:
  - David Mathog, Caltech

This work was supported in part by the U.S. Department of Energy under the ASCR X-Stack program (contract DESC0005380) and contract DE-AC02-06CH11357.

Questions?

- Try Swift/T:
  - http://www.mcs.anl.gov/exm/local/guides/swift.html
- My SC12 talk:
  - "Design and Analysis of Data Management in Scalable Parallel Scripting", 1:30PM - 2:00PM, Thursday, Nov 15th, 255-EF
- Speaker contact:
  - zhaozhang@uchicago.edu
  - http://www.ci.uchicago.edu/~zzhang