What do they have in common?

Windows Azure Storage

A Highly Available Cloud Storage Service with Strong Consistency

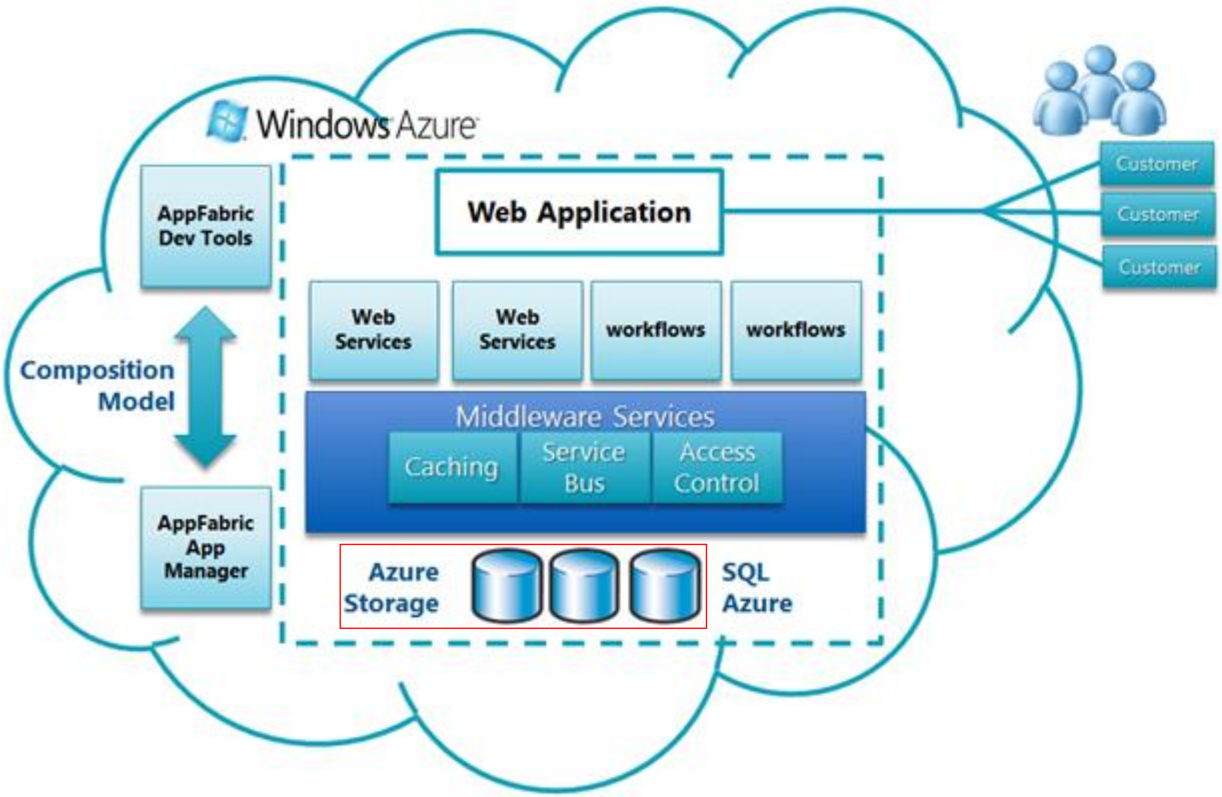
Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, Leonidas Rigas @ Microsoft

Presenter: Lionel Li

Key Features

- Strong Consistency and Highly Available
- Durability
- Global and Scalable Namespace/Storage
  - Consistent accessibility anywhere
  - Address exabytes of data and beyond
- Disaster Recovery
- Multi-Tenancy and Cost of Storage
  - Multiple customers served from the same shared storage

Windows Azure Cloud Platform



Source: http://blogs.msdn.com/

WAS Fundamentals

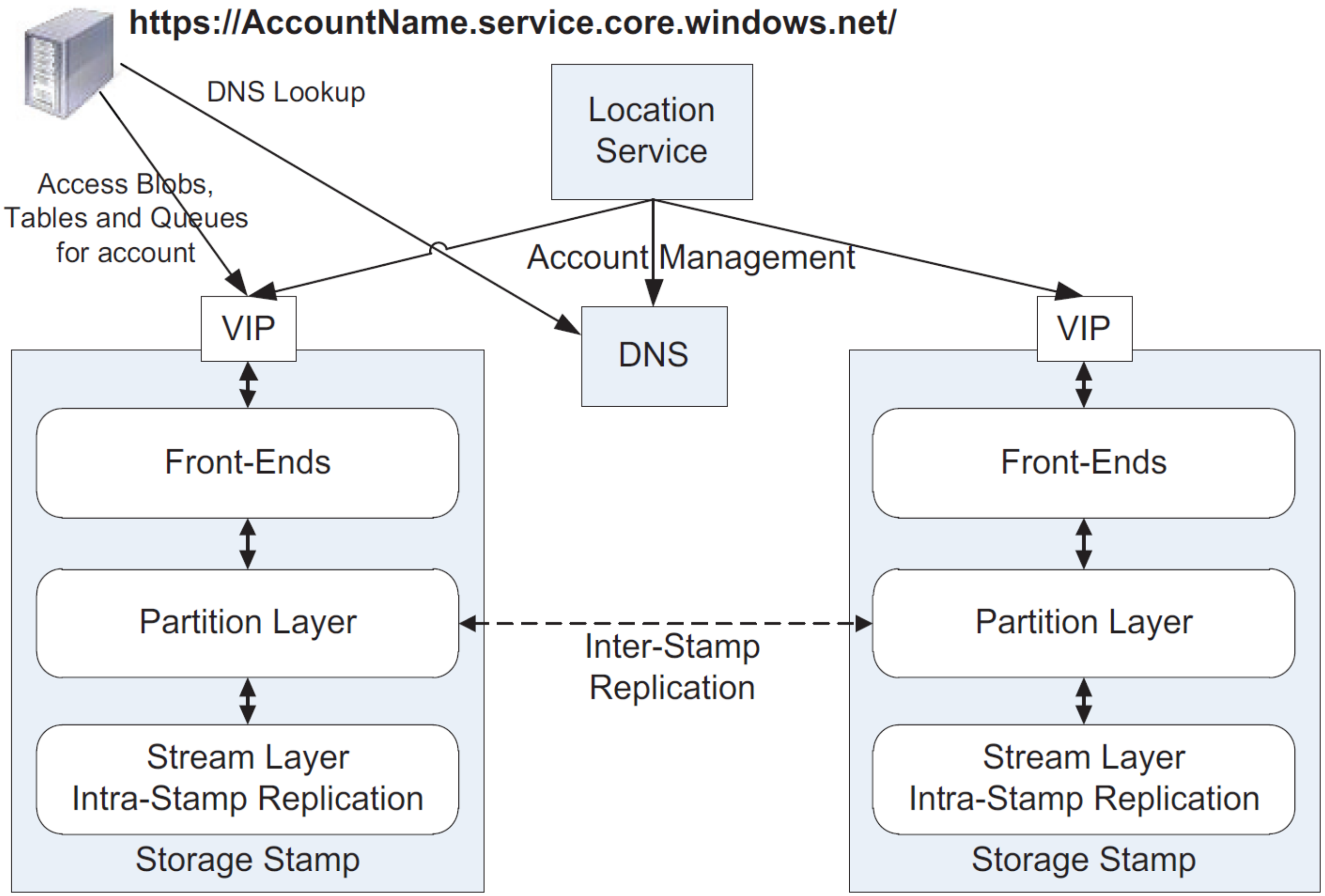
Global Partitioned Namespace

http(s)://**AccountName**.<service>.core.windows.net/**PartitionName**/**ObjectName**

Storage Abstractions

- Blob (Binary Large Object)
  - Named files along with metadata for the file
  - http://BobsDVD.blob.core.windows.net/Comedy/FunnyMovie.mp4
- Tables
  - Highly scalable non-relational database
  - http://sally.table.core.windows.net/Customer
- Queues
  - Reliable storage and delivery of messages

General Architecture

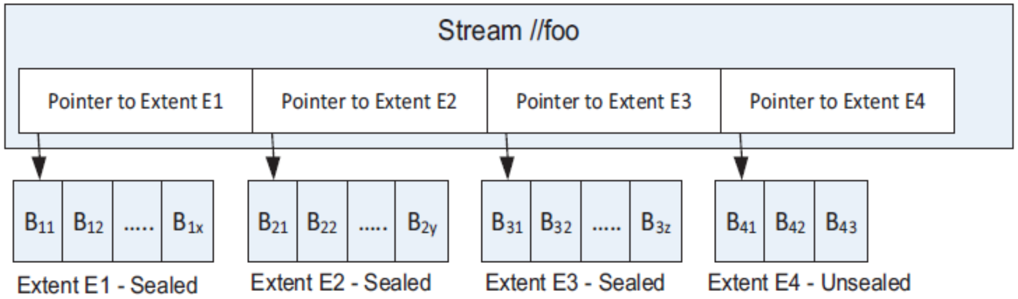

https://AccountName.service.core.windows.net/
DNS Lookup
Location Service
Access Blobs, Tables and Queues for account
Account Management
VIP
DNS
VIP
Front-Ends
Partition Layer
Stream Layer Intra-Stamp Replication
Storage Stamp
Inter-Stamp Replication
Front-Ends
Partition Layer
Stream Layer Intra-Stamp Replication
Storage Stamp

Location Service

- Manages all storage stamp
- Manages account namespace
  - Across storage stamps
  - Disaster Recovery
  - Load Balancing
- Scales additional location and storage
- Respects Location Affinity
- Updates DNS
- Durable – Distributed across 2 geo locations

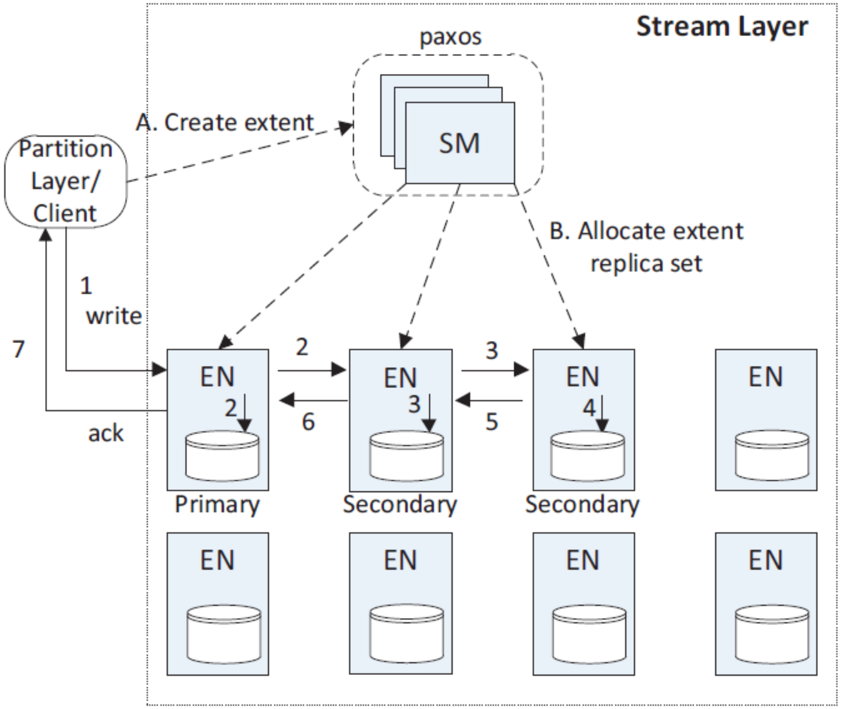
Stream Layer – Fundamentals

- Block – min unit of read/write
  - Variable in size (up to 4MB)
  - Checksum performed
- Extent – unit of replication
  - Sealing
  - Size limit of 1GB
- Stream – Ordered list of pointers to extents
  - Hierarchical namespace
  - Append-only



Stream Layer – Architecture

- An *append-only* DFS
- Data stored as files
- Replication factor of 3
- Re-replicate on:
  - EN failure
  - Disk failure
  - Checksum mismatch

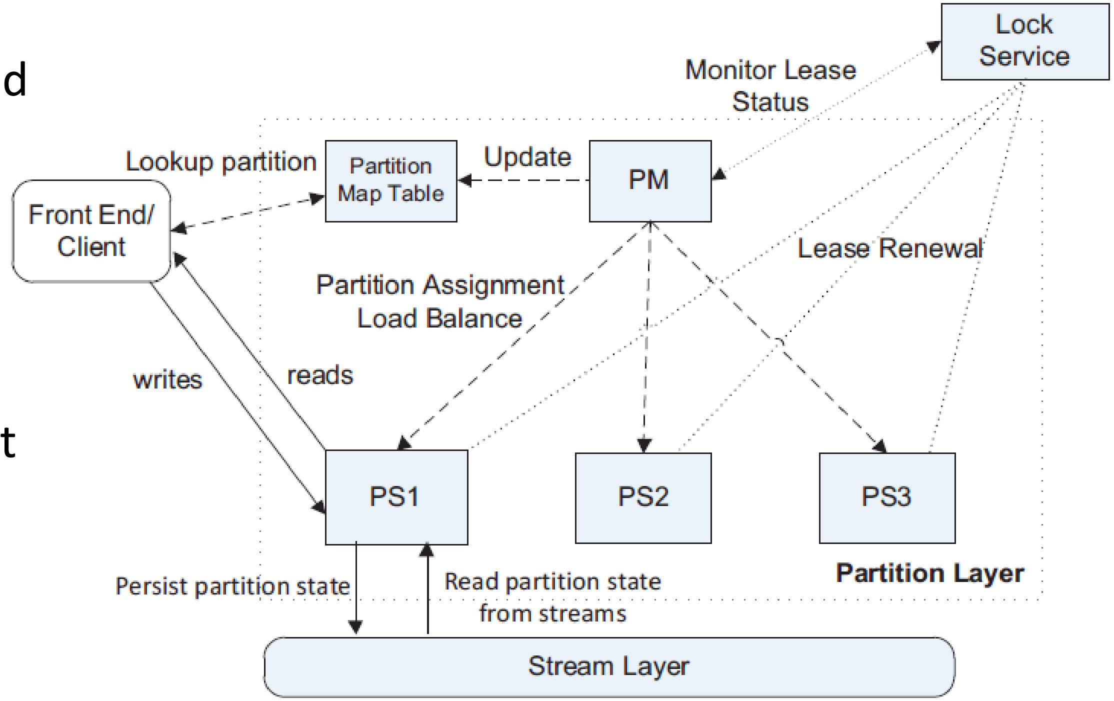


Stream Layer – Load Balancing

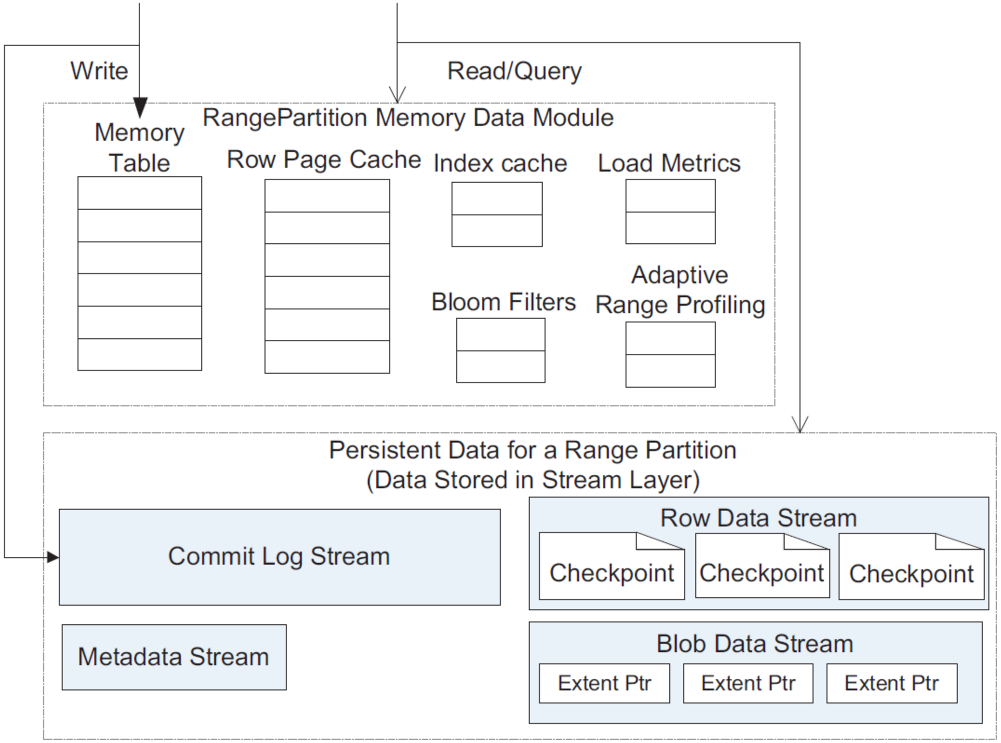
- Deadline
  - Read requests are submitted with a “deadline” requirement
  - EN replies to client that deadline cannot be met if it determines that it cannot meet the deadline
- Sealing
  - Sealed extents are bitwise identical
- Erasure Coding (Reed-Solomon)
  - Improves durability
  - Data compression from 3x to 1.3x -1.5x

Partition Layer – Architecture

- Object Tables
    - Maintained for each data abstraction
    - RangePartitions assigned across PS
- Partition Manager
    - Several in a stamp
    - Heartbeats to PS
- Partition Server
    - Servers requests to a set of RangePartitions
- Partition Map
    - Maps RangePartition to corresponding PS
    - Used by FE for request routing



Partition Layer - RangePartition



Partition Layer – Load Balancing

- Reassign RangePartition
    - PS has too high work load evenly spread out across RangePartition
- Split
    - Load-based or Size-based
- Merge
    - To keep # of RangePartitions proportional to # of PS in a stamp

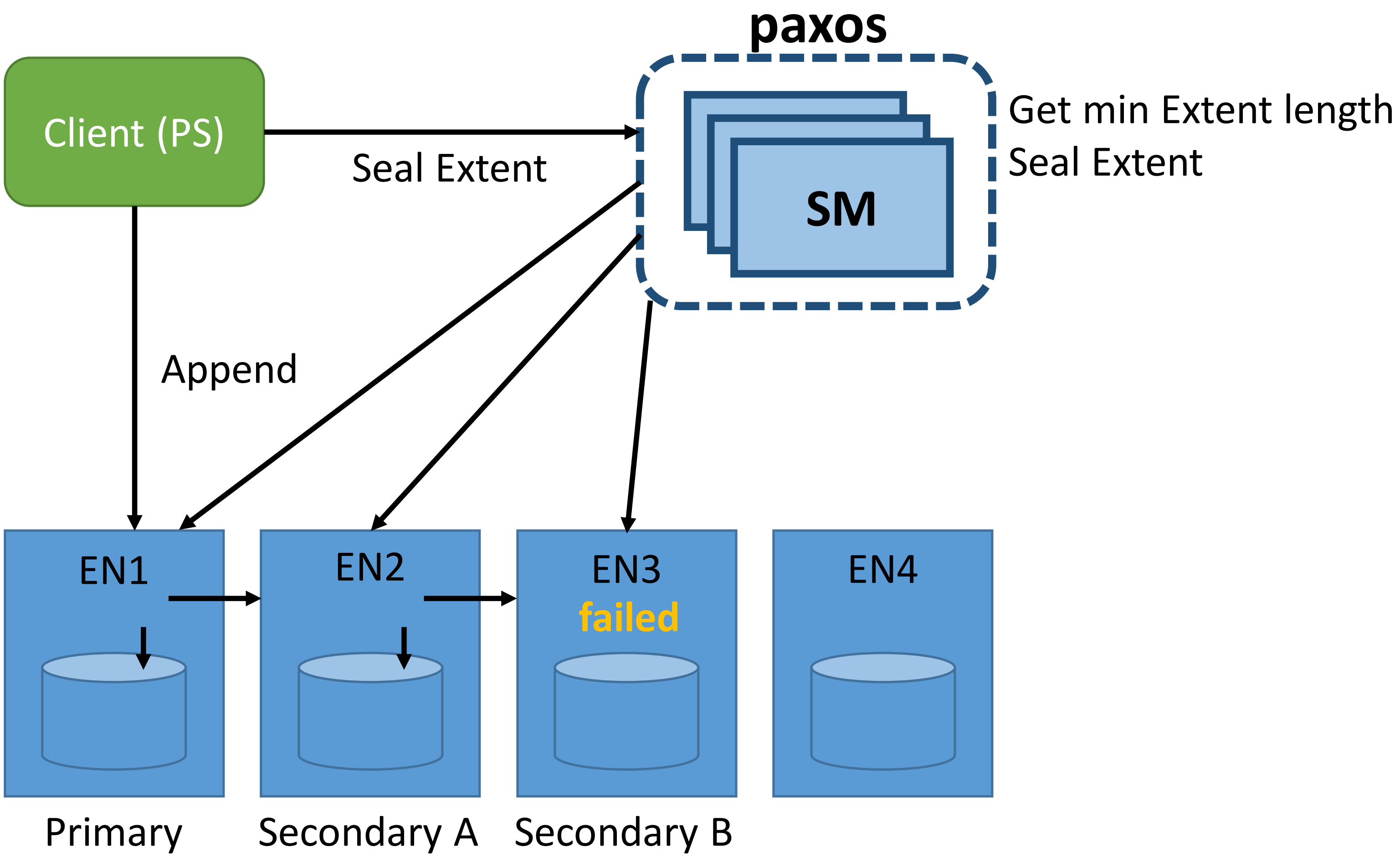
Load Balancing

- Location Service
  - Allocates accounts to storage stamps and manages them across storage stamps
  - Distributed across 2 geographic locations
- Stream Layer
  - EN serves a read request only if it can meet deadline
- Partition Layer
  - RangePartition Load Balancing – Reassigning, Splitting and Merging

Replication for Consistency

- Intra-Stamp Replication
  - Synchronous replication within stamp
  - Critical path of customer’s write request
  - Durability against hardware failures
- Inter-Stamp Replication
  - Asynchronous replication (done in background)
  - Geo-redundancy against natural disasters within a geographical location

Failure Handling



Throughput

- Random 1KB get/put requests
- Against 100GB Table

- Random blob get/put requests
- 4MB blobs per request

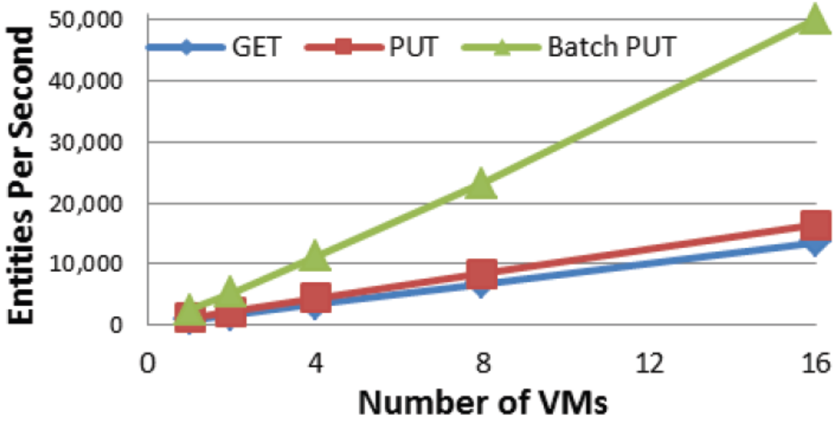


Figure 6 Table Entity Throughput for 1-16 VMs

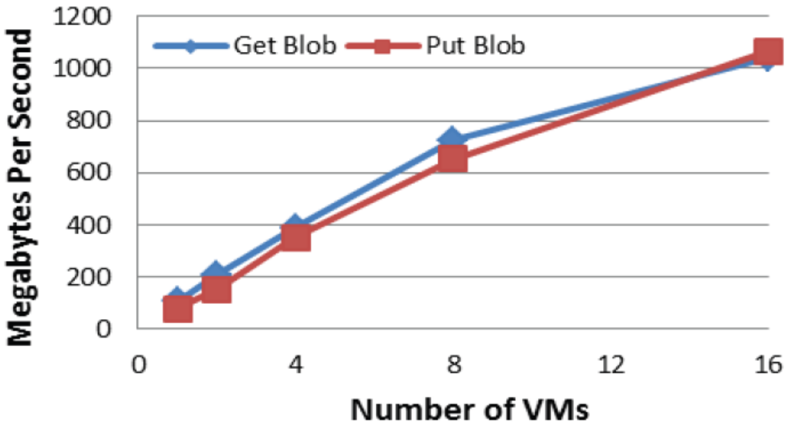
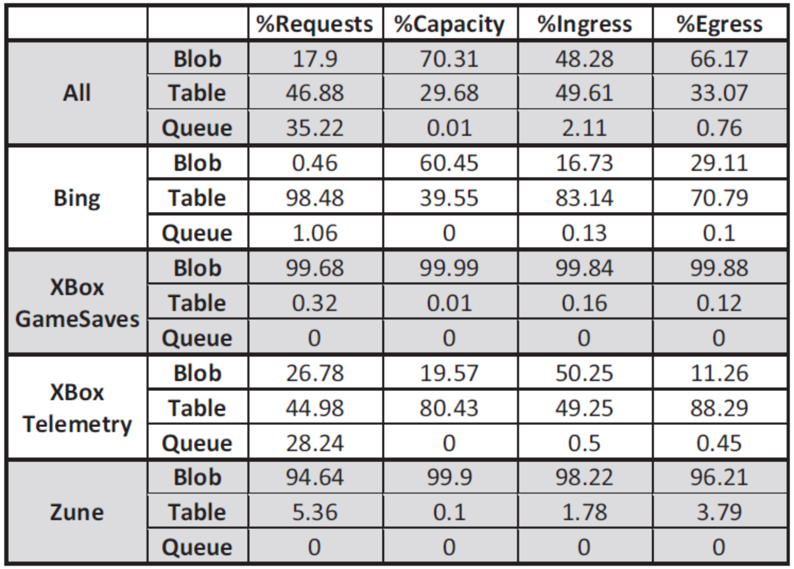


Figure 7: Blob Throughput for 1-16 VMs

Workload Profiles

Table 1: Usage Comparison for (Blob/Table/Queue)



		%Requests	%Capacity	%Ingress	%Egress
All	Blob	17.9	70.31	48.28	66.17
	Table	46.88	29.68	49.61	33.07
	Queue	35.22	0.01	2.11	0.76
Bing	Blob	0.46	60.45	16.73	29.11
	Table	98.48	39.55	83.14	70.79
	Queue	1.06	0	0.13	0.1
XBox GameSaves	Blob	99.68	99.99	99.84	99.88
	Table	0.32	0.01	0.16	0.12
	Queue	0	0	0	0
XBox Telemetry	Blob	26.78	19.57	50.25	11.26
	Table	44.98	80.43	49.25	88.29
	Queue	28.24	0	0.5	0.45
Zune	Blob	94.64	99.9	98.22	96.21
	Table	5.36	0.1	1.78	3.79
	Queue	0	0	0	0

Discussions/Comments

- Paper is more of an engineering effort than introducing new concepts to the field
- Differences between Windows Azure Storage vs Spanner, Bigtable or AWS as a storage system
- What are the ways in which client deals with duplicate records? The solution seems to provide an append-only solution at stream level. Are no block deletions supported at all? Or will the extent save information about valid blocks and invalid blocks?
- Still unclear if CAP Theorem was really violated. Synchronous replication is on the critical path for all customer write requests, so it must happen before the write is considered complete. Maybe if you partition the network just right, the synchronous replication guarantee could be exploited to weaken the availability guarantee.
- One major limitation of the WAS is that it is very difficult to port applications that are not windows centric to run on the WAS because it needs Microsoft workflow. And no options to install operation systems other than Windows is allowed in WAS

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

Authors: Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica

Presented By: Hilfi Alkaff

Background

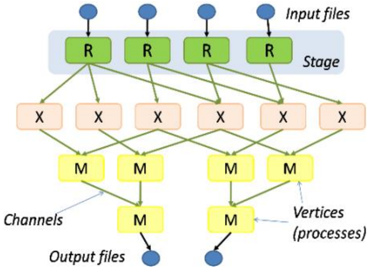
- Gazillions of cluster computing frameworks

hadoop MapReduce

Google Pregel

Pig

RAILS



Dryad

CIEL

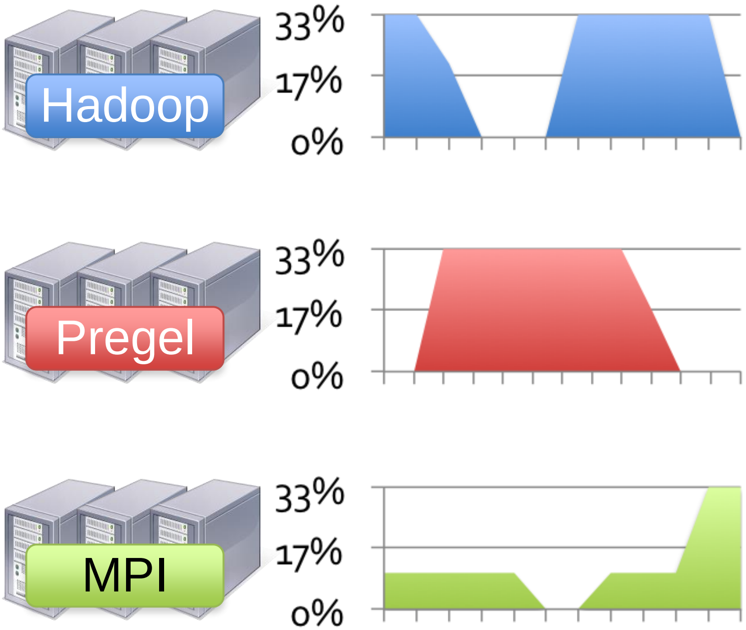
S4 distributed stream computing platform

Google Percolator

MPI-2

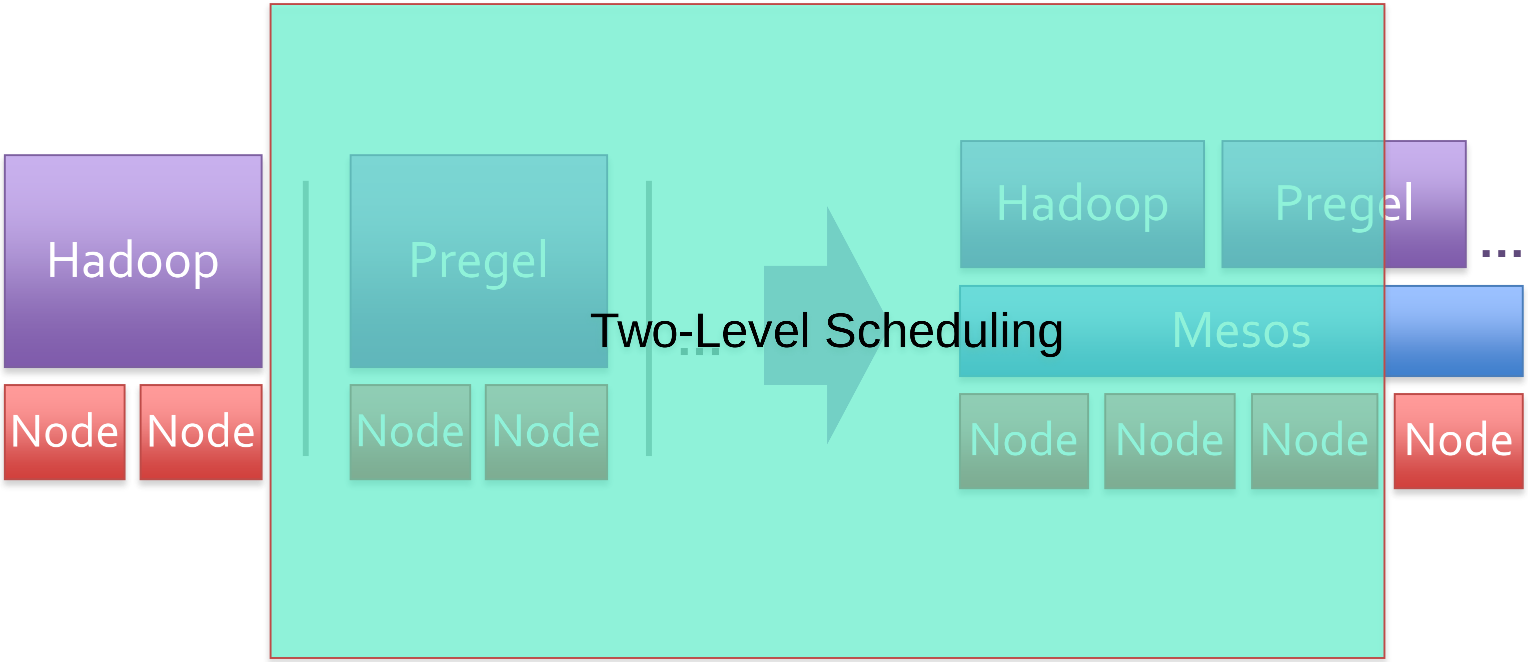
One Framework per Cluster

- Inefficient resource usage
- Hard to share data
- Hard to cooperate



One Framework to Rule Them All

- Common resource sharing layer over which diverse frameworks can run



Side Benefits

- Run multiple instances of the *same* framework
- Build specialized frameworks targeting particular problem domains

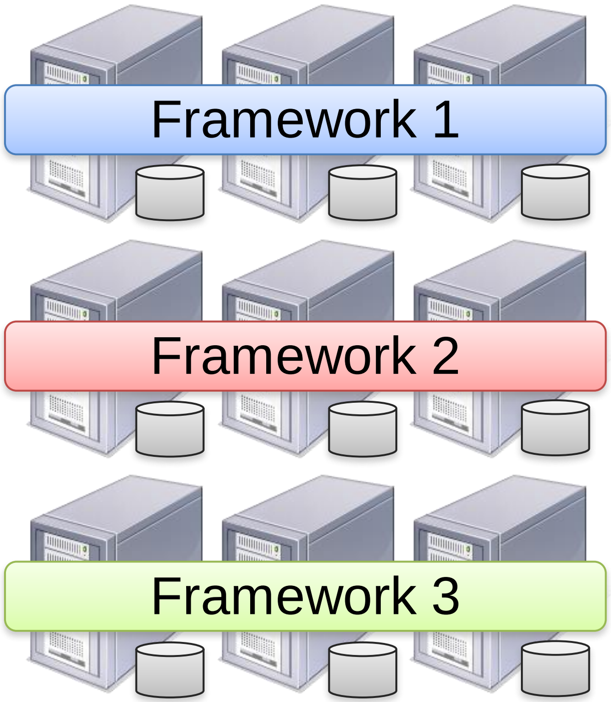
Objectives

- High utilization of resources
- Diverse frameworks (Current and Future)
- Scalability
- Reliability to machine failures

Design Elements

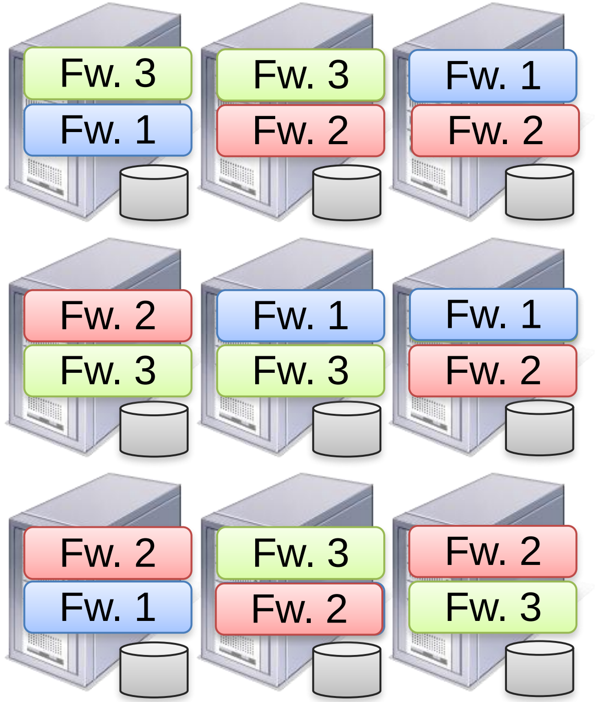
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):



Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):



Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality

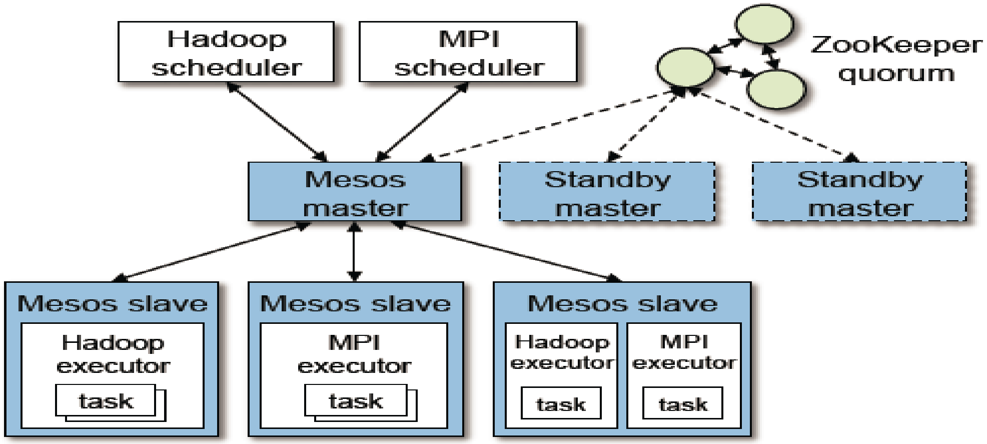
Element 2: Resource Offers

Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

+ Keeps Mesos simple, lets it support future frameworks

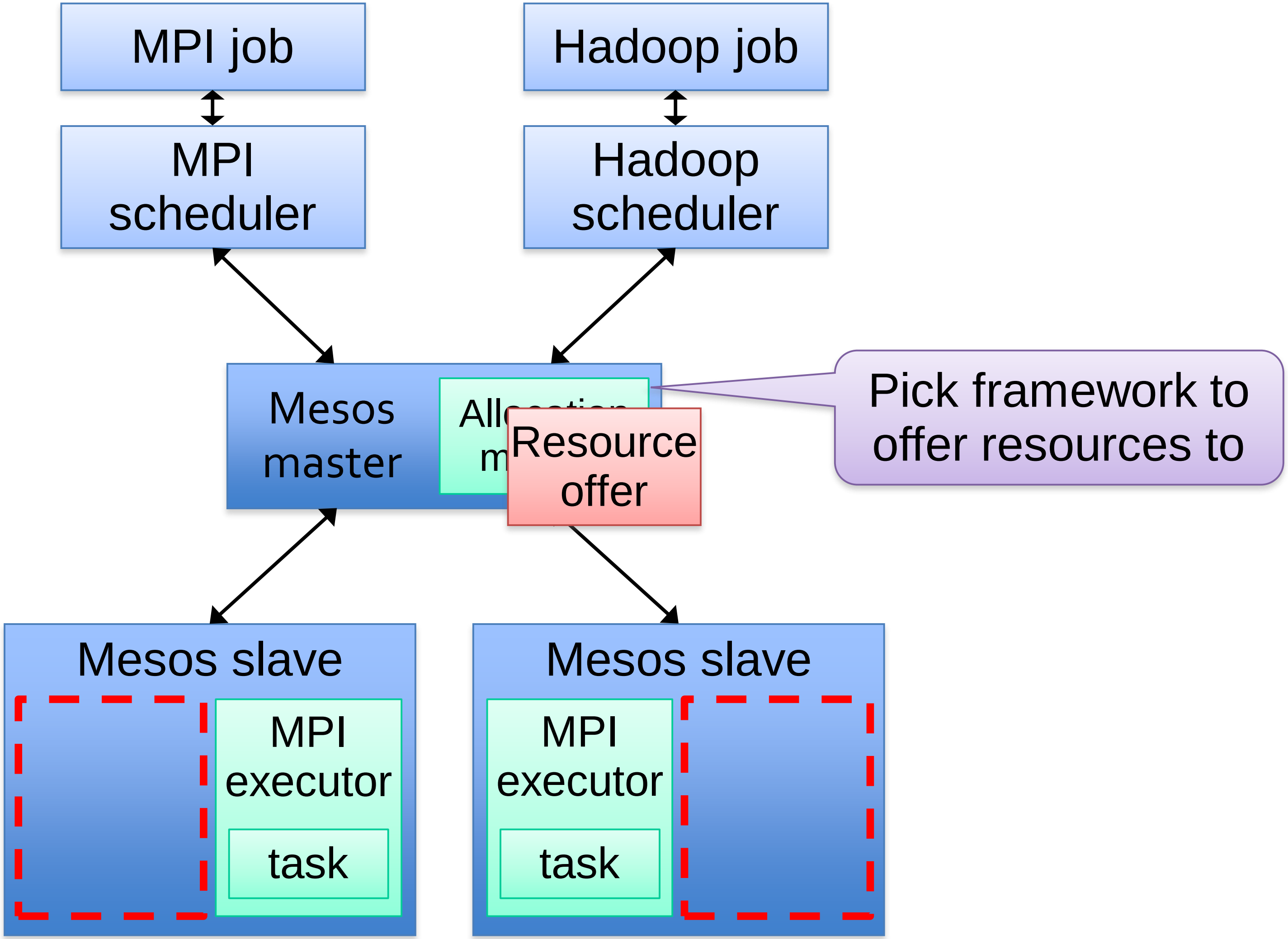
- Decentralized decisions might not be optimal

Architecture

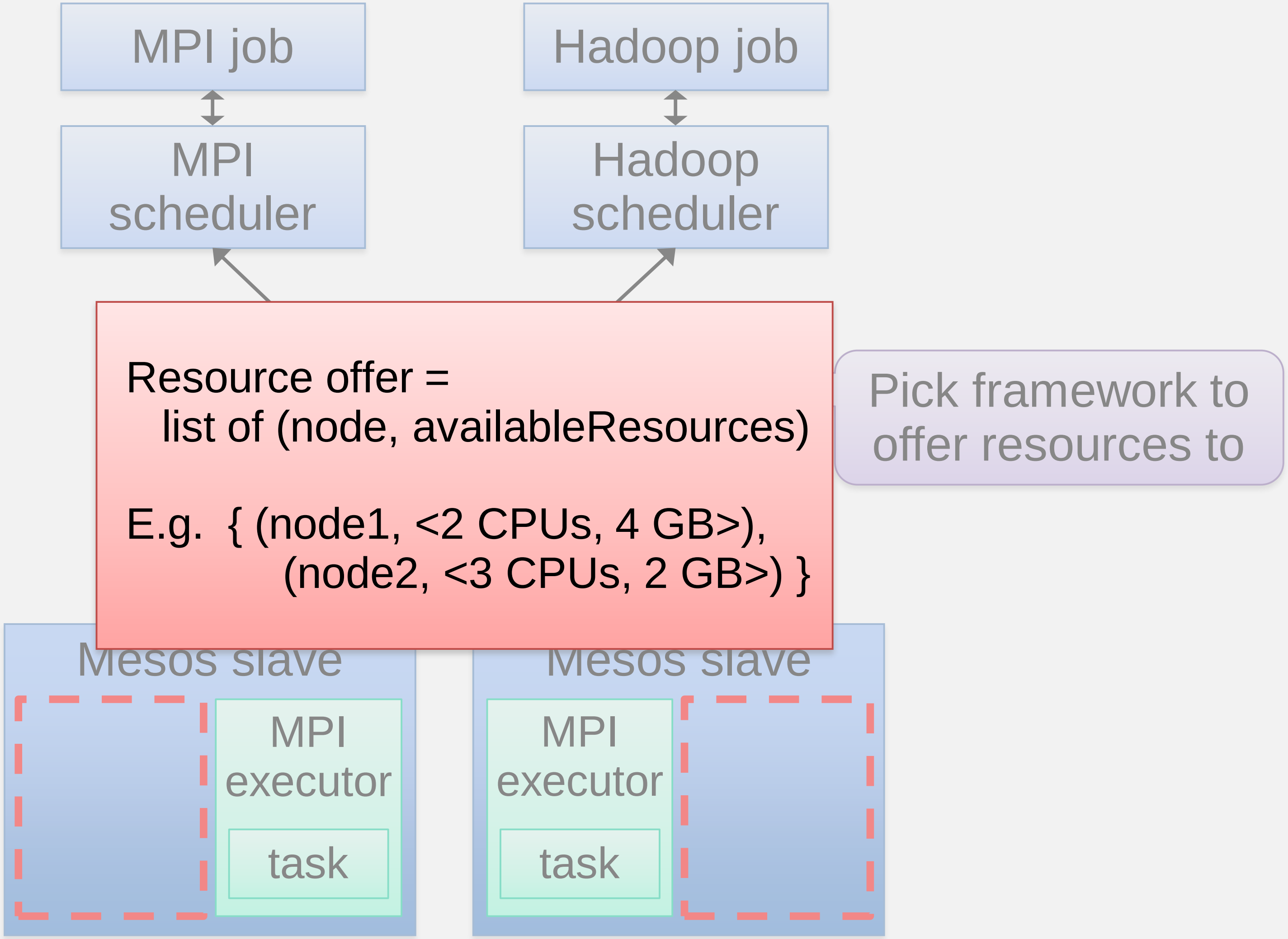


- Master: Implements fine-grained sharing across frameworks using resource offers
- Scheduler: Registers with the master and select which resources to accept from the master
- Executor: Launched on slave nodes to run the framework’s tasks
- Slave: It’s a slave

Event Flow


MPI job
Hadoop job
MPI scheduler
Hadoop scheduler
Mesos master
Allocation m
Resource offer
Pick framework to offer resources to
Mesos slave
MPI executor
task
Mesos slave
MPI executor
task

Event Flow

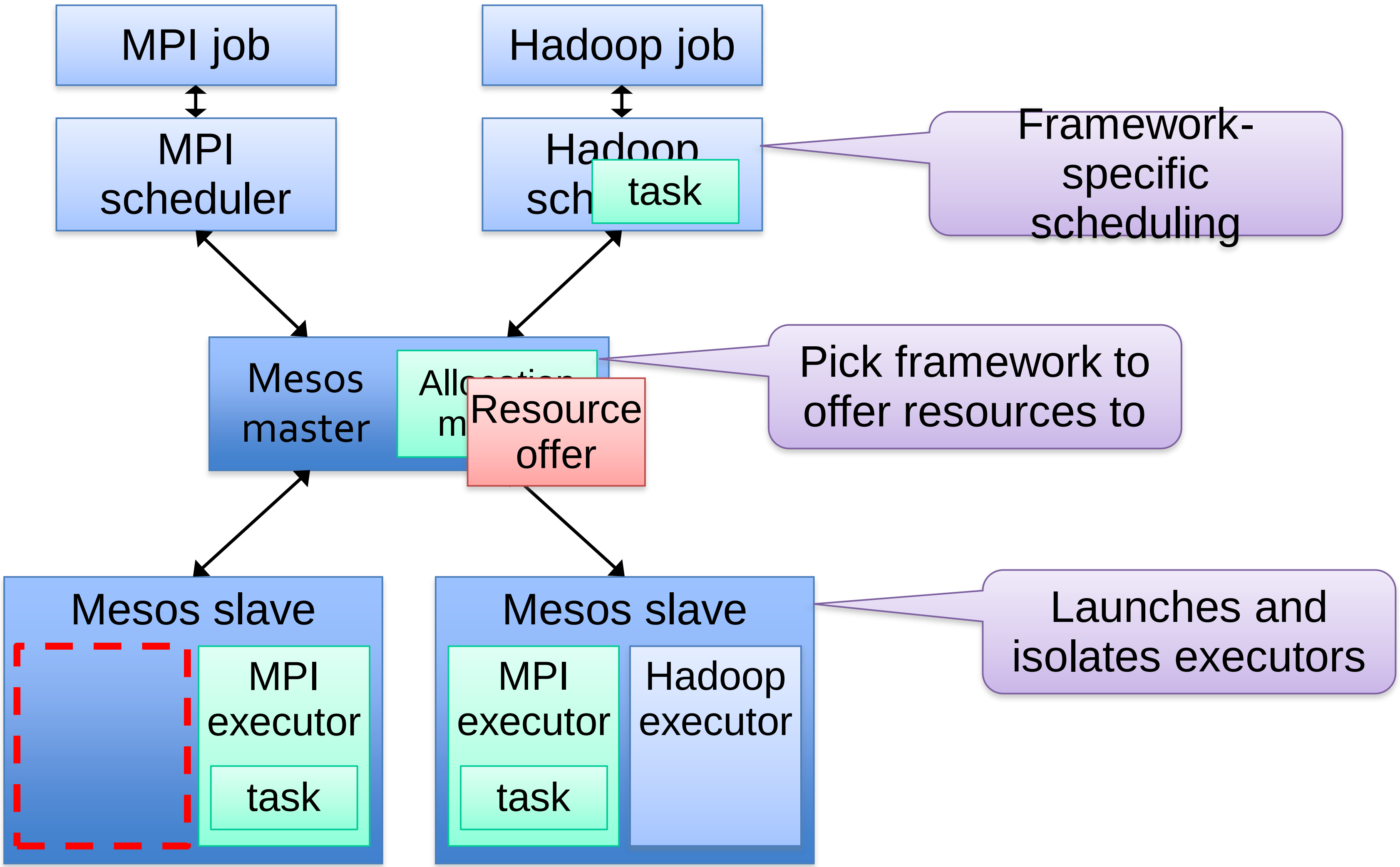


Resource offer =
list of (node, availableResources)

E.g. { (node1, <2 CPUs, 4 GB>),
(node2, <3 CPUs, 2 GB>) }

Pick framework to offer resources to

Event Flow



MPI job

Hadoop job

MPI scheduler

Hadoop sch

task

Framework-specific scheduling

Mesos master

Allocation m

Resource offer

Pick framework to offer resources to

Mesos slave

MPI executor

task

Mesos slave

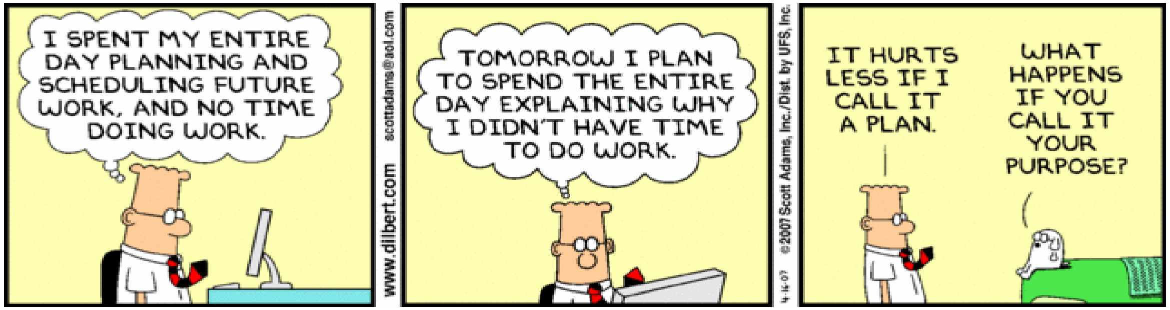
MPI executor

task

Hadoop executor

Launches and isolates executors

Scheduling

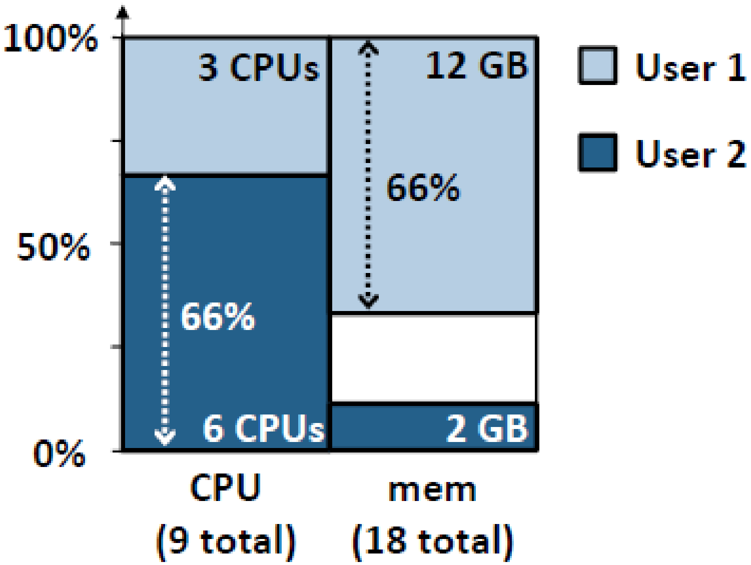

I SPENT MY ENTIRE DAY PLANNING AND SCHEDULING FUTURE WORK, AND NO TIME DOING WORK.
www.dilbert.com scottadams@aol.com
TOMORROW I PLAN TO SPEND THE ENTIRE DAY EXPLAINING WHY I DIDN'T HAVE TIME TO DO WORK.
4-16-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.
IT HURTS LESS IF I CALL IT A PLAN.
WHAT HAPPENS IF YOU CALL IT YOUR PURPOSE?

- Max-min fairness
- Strict priorities
- **Domain Resource Fairness [NSDI '11]

Digression: Domain Resource Fairness

- Idea: Apply max-min fairness to dominant shares
- Schedule a task with the smallest dominant share
  - O(log n) time per decision

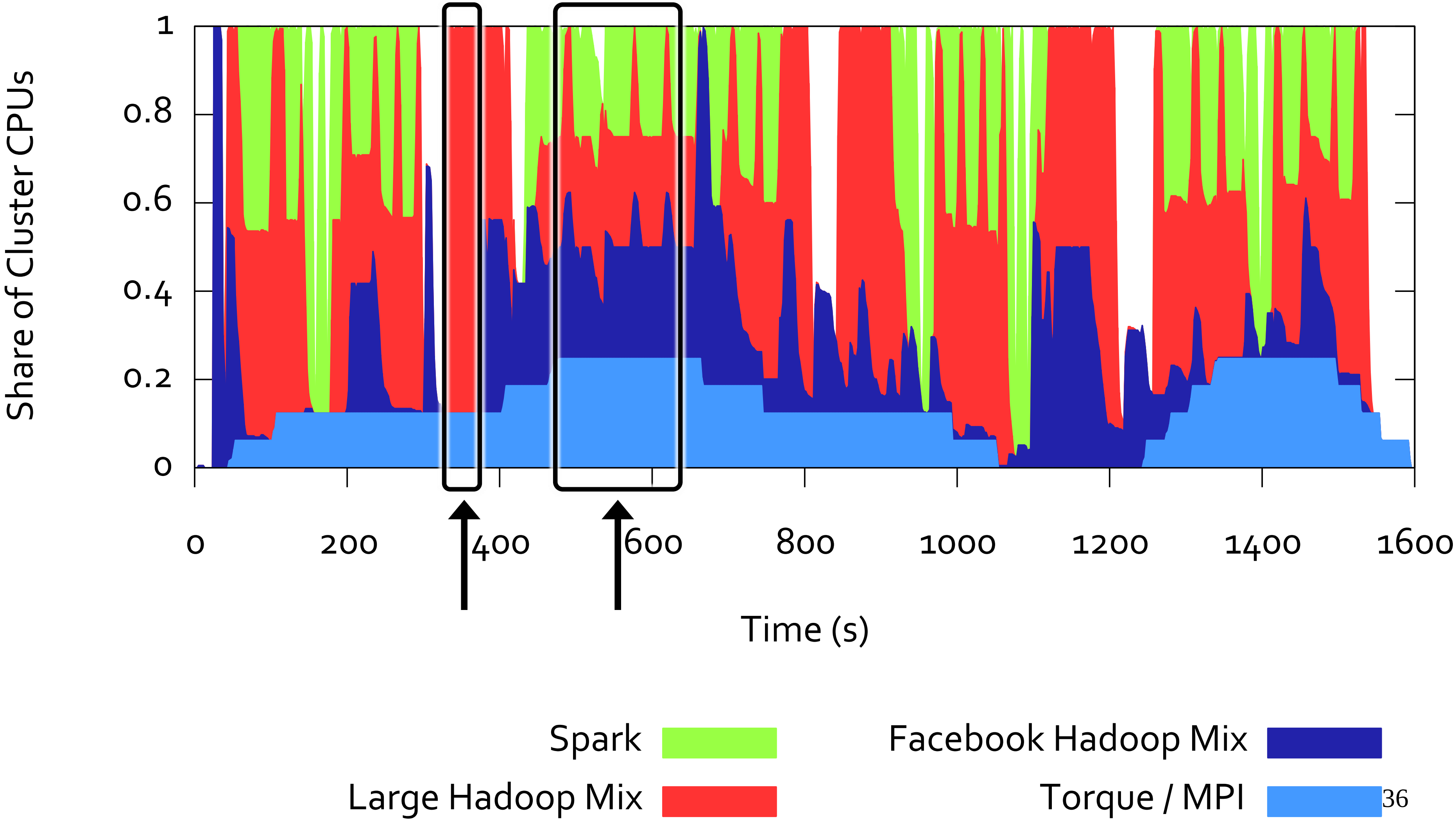
- Example:
  - Total Resources: <9CPU, 18GB>
  - User 1 demand: <1 CPU, 4 GB>
  - User 2 demand: <3 CPU, 1 GB>



Fun Facts

- 20k C++
- Frameworks supported: Hadoop, MPI, Torque, Spark, HyperTable
- Master fail-over using ZooKeeper
- Isolation using LXC
- Users: Twitter, ML researchers @ Berkeley, Conviva, UCSF
- Java, Python, C++ API

Dynamic Resource Sharing



Mesos vs Static Partitioning

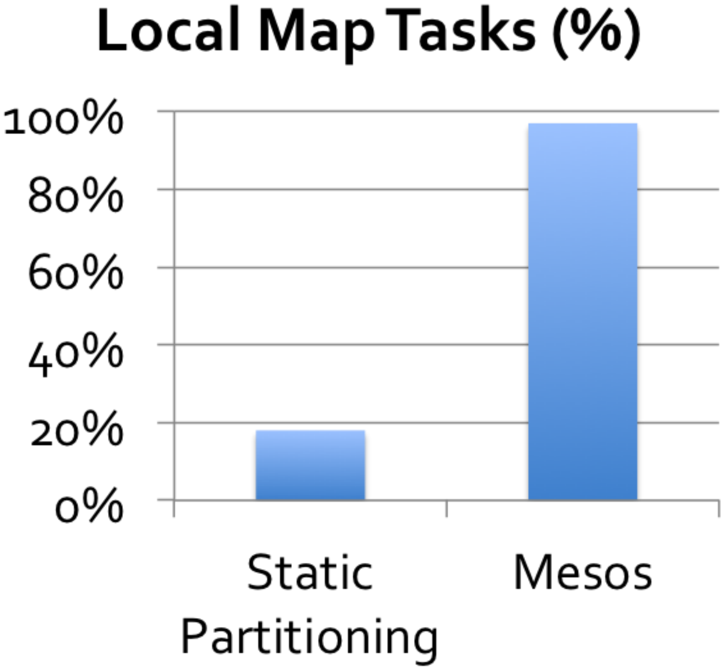
Compared performance with statically partitioned cluster where each framework gets 25% of nodes

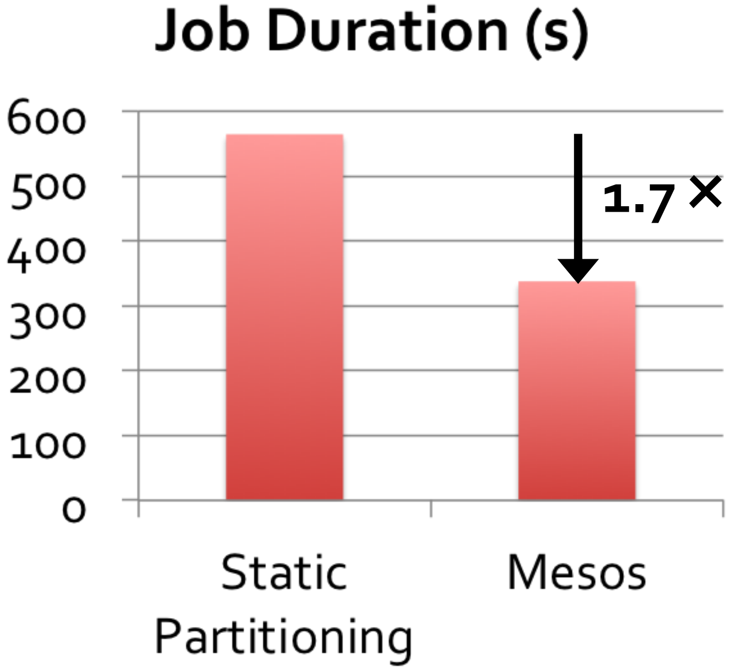
Framework	Speedup on Mesos
Facebook Hadoop Mix	1.14 ×
Large Hadoop Mix	2.10 ×
Spark	1.26 ×
Torque / MPI	0.96 ×

Data Locality with Resource Offers

Ran 16 instances of Hadoop on a shared HDFS cluster

Used delay scheduling [EuroSys '10] in Hadoop to get locality (wait a short time to acquire data-local nodes)



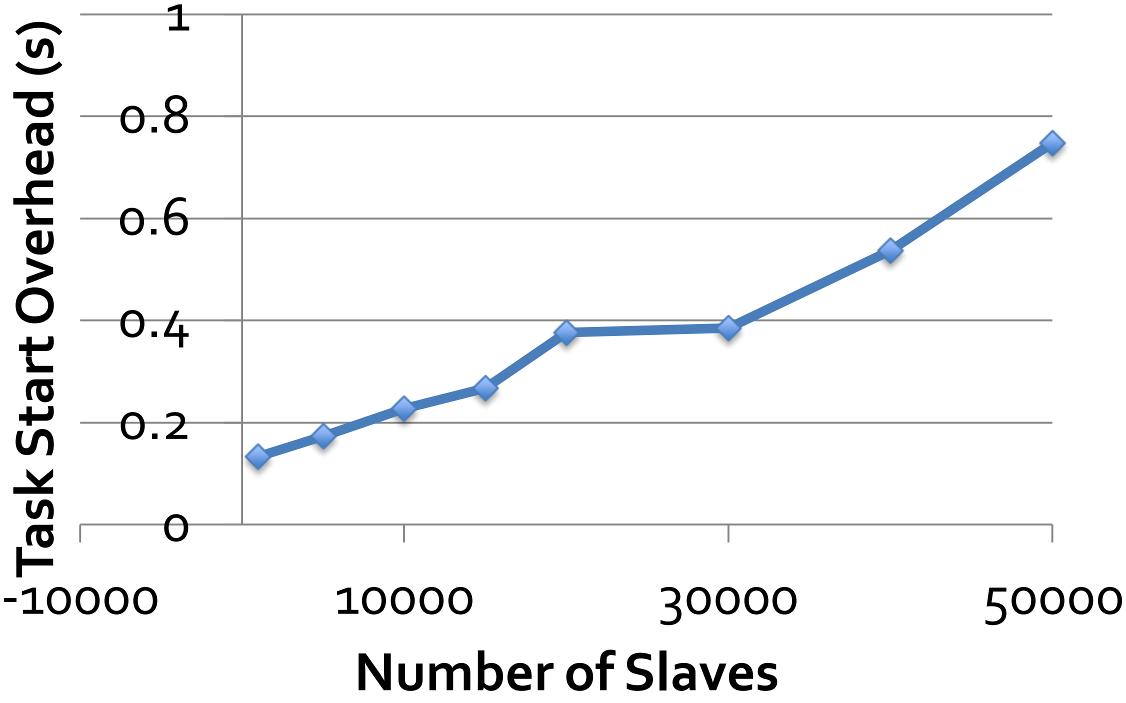


Scalability

Mesos only performs *inter-framework* scheduling (e.g. fair sharing), which is easier than intra-framework scheduling

Result:

Scaled to 50,000 emulated slaves, 200 frameworks, 100K tasks (30s len)



Fault Tolerance

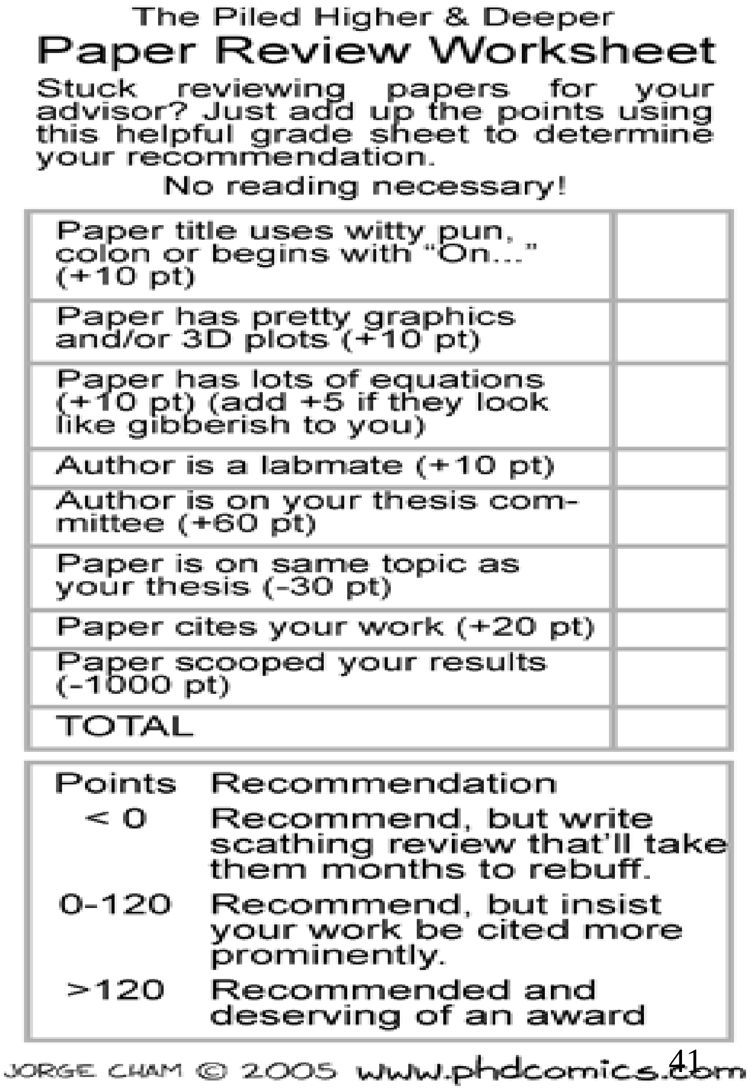
Mesos master has only *soft state*: list of currently running frameworks and tasks

Rebuild when frameworks and slaves re-register with new master after a failure

Result: fault detection and recovery in ~10 sec

Review

- Shares clusters efficiently among diverse framework
- Very suited to short tasks
- Applies to heterogeneous nodes
- Incentive-compatible



The Piled Higher & Deeper

Paper Review Worksheet

Stuck reviewing papers for your advisor? Just add up the points using this helpful grade sheet to determine your recommendation.

No reading necessary!

Paper title uses witty pun, colon or begins with "On..." (+10 pt)	
Paper has pretty graphics and/or 3D plots (+10 pt)	
Paper has lots of equations (+10 pt) (add +5 if they look like gibberish to you)	
Author is a labmate (+10 pt)	
Author is on your thesis committee (+60 pt)	
Paper is on same topic as your thesis (-30 pt)	
Paper cites your work (+20 pt)	
Paper scooped your results (-1000 pt)	
TOTAL	

Points	Recommendation
< 0	Recommend, but write scathing review that'll take them months to rebuff.
0-120	Recommend, but insist your work be cited more prominently.
>120	Recommended and deserving of an award

JORGE CHAM © 2005 www.phdcomics.com

Thoughts

- The “Mesos” programming language?
- Utilizing Software-Defined Networking
- Handling revocation cleaner?
  - There are still wasted bandwidths
- Shady fault tolerance mechanism
  - Is using soft-state good?
  - How good does it scale?

Untangling Cluster Management with Helix

Kishore Gopalakrishna, Shi Lu, Zhen Zhang, Adam Silberstein, Kapil Surlaker,Ramesh Subramonian, Bob Schulman @ LinkedIn

What is Helix?

- Generic cluster management framework
- Automatic management of partitioned, replicated, distributed resources hosted on cluster of nodes

Why Helix?

- Abstract cluster management from the core functionality.
- Quick transformation from a single node system to a distributed system.
- Two level scheduling.
- Since the controllers goal is to satisfy state machine constraints at all times, use cases like cluster startup, node failure, cluster expansion are solved in a similar way

Cluster Management Comparisons

	Windows Azure Storage	Mesos	Helix
Node failure detection & recovery	Yes	Yes	Yes
Dynamic addition of nodes	Yes	Yes	Yes
Auto load-balancing	Yes	Yes	Yes
Define custom behavior & constraints	No	No	Yes
Multiple Masters	No	No	Yes