Clustera: A data-centric approach to scalable cluster management

David J. DeWitt
Eric Robinson
Srinath Shankar
Erik Paulson
Jeff Naughton
Andrew Krioukov
Joshua Royalty

Computer Sciences Department
University of Wisconsin-Madison

Outline

- A historical perspective
- A taxonomy of current cluster management systems
- Clustera - the first DBMS-centric cluster management system
- Examples and experimental results
- Wrapup and summary

A Historical Perspective

- Concept of a “cluster” seems to have originated with Wilke’s idea of “Processor bank” in 1980
- “Remote Unix” (RU) project at Wisconsin in 1984
  - Ran on a cluster of 20 VAX 11/750s
  - Supported remote execution of jobs
  - I/O calls redirected to submitting machine
- “RU” became Condor in late 1980s (Livny)
  - Job checkpointing
  - Support for non-dedicated machines (e.g. workstations)
  - Today, deployed on 1500+ clusters and 100K+ machines worldwide (biggest clusters of 8000-15000 nodes)

No, Google did not invent clusters

- Cluster of 20 VAX 11/750s circa 1985 (Univ. Wisconsin)

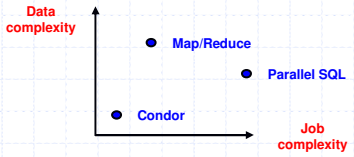
Clusters and Parallel DB Systems

- Gamma and RU/Condor projects started at the same time using same hardware. Different focuses:
- RU/Condor:
  - Computationally intensive jobs, minimal I/O
  - "High throughput" computing
- Gamma
  - Parallel execution of SQL
  - Data intensive jobs and complex queries
- Competing parallel programming efforts (e.g. Fortran D) were a total failure
  - Probably why Map-Reduce is so "hot" today

What is a cluster management system?

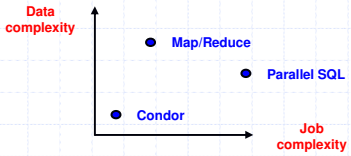
- Provide simplified access for executing jobs on a collection of machines
- Three basic steps:
  - Users submit jobs
  - System schedules jobs for execution
  - Run jobs
- Key services provided:
  - Job queuing, monitoring
  - Job scheduling, prioritization
  - Machine management and monitoring

Condor



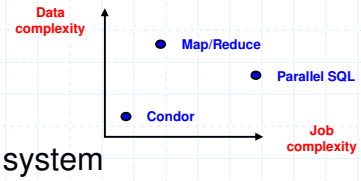
- Simple, computationally intensive jobs
  - Complex workflows handled outside the system
- Files staged in and out as needed
  - Partially a historical artifact and desire to handle arbitrary sized data sets
- Scheduler pushes jobs to machines based on a combination of priorities and fair share scheduling
- Tons of other features including master-worker, glide-in, flocking of pools together, …

Parallel SQL



- Tables partitioned across nodes/disks using hash or range partitioning
  - No parallel file system
- Optimizer: SQL query ==> query plan (tree of operators)
- Job scheduler: parallelizes query plan
- Scalability to 1000s of nodes
- Failures handled using replication and transactions
- All key technical details worked out by late 1980s

Map/Reduce



- Files stored in distributed file system
  - Partitioned by chunk across nodes/disks
- Jobs consist of a Map/Reduce pair
  - Each Map task:
    - Scans its piece of input file, producing output records
    - Output records partitioned into M local files by hashing on output key
  - Each Reduce task:
    - Pulls N input files (one from each map node)
    - Groups of records with same key reduced to single output record
- Job manager:
  - Start and monitor N map tasks on N nodes
  - Start and monitor M reduce tasks on M nodes

Summary

- All three types of systems have distinct notions of jobs, files, and scheduler
- It is definitely a myth MR scales better than parallel SQL
  - See upcoming benchmark paper
- MR indeed does a better a job of handling failures during execution of a job

The Big Question

- Seem to be at least three distinct types of cluster management systems
- Is a unified framework feasible?
- If so, what is the best way of architecting it?
- What is the performance penalty?

Outline

- A historical perspective
- A taxonomy of current cluster management systems
- Clustera – a DBMS-centric cluster management system
- Examples and experimental results
- Wrapup and summary

Clustera Project Goals

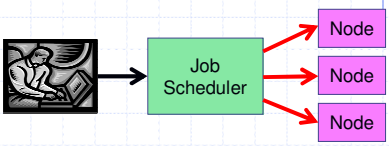
- Leverage modern, commodity software including relational DB systems and application servers such as Apache Jboss
- Architecturally extensible framework
  - Make it possible to instantiate a wide range of different types of cluster management systems (Condor, MR, parallel SQL)
- Scalability to thousands of nodes
- Tolerant to hardware and software failures

Why cluster management is a DB problem

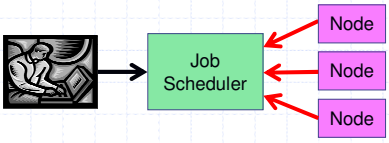
- Persistent data
  - The job queue must survive a crash
  - Accounting information must survive a crash
  - Information about nodes, files, and users must survive a crash
- Transactions
  - Submitted jobs must not be lost
  - Completed jobs must not reappear
  - Machine usage must be accounted for
- Query processing
  - Users need to monitor their jobs
  - Administrators need to monitor system health

Push vs. Pull

- Push
  - Jobs pushed to idle nodes by job scheduler
  - Standard approach: Condor, LSF, MR, parallel DB systems

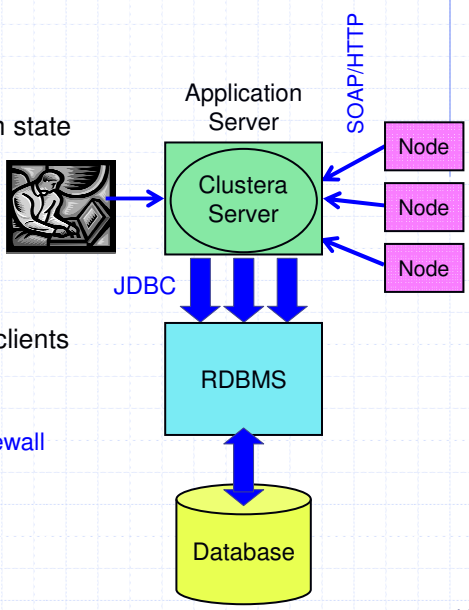


- Pull
  - Idle nodes pull jobs from job scheduler
  - Trivial difference but truly simpler as job scheduler becomes purely a server
  - Allows Clustera to leverage application server technology



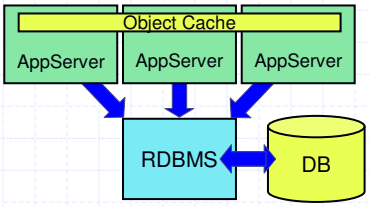
Clustera Architecture

- RDBMS used to hold all system state
- All cluster logic runs in the application server (e.g. JBoss)
  - Job mgmt. and scheduling
  - Node management
  - File management
- Nodes are simply web-service clients of the app. server
  - Used to run jobs
  - Require a single hole in the firewall



Why??

- Use of RDBMS should be obvious
- Why an Application Server?
  - Proven scalability to 10s of 1000s of web clients
  - Multithreaded, scalable, and fault tolerant



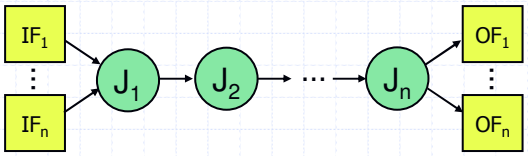
  - Pooling of connections to DBMS
  - Portability (Jboss, Websphere, WebLogic, …)
    - Also hides DBMS specific features

Basis of Clustera Extensibility

- Four key mechanisms
  - Concrete Jobs
  - Concrete Files
  - Logical files and relational tables
  - Abstract jobs and abstract job scheduler

Concrete Jobs

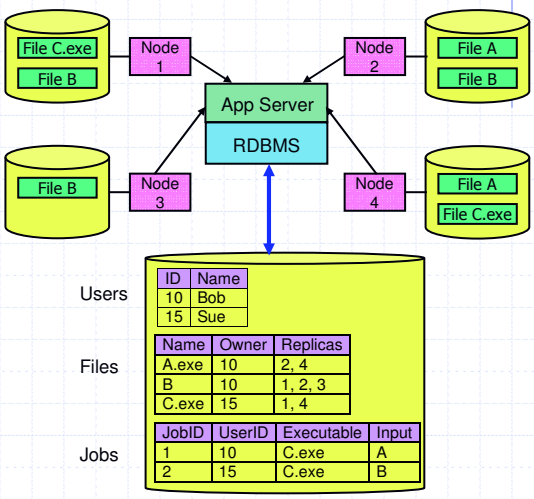
- Pipeline of executables with zero or more input and output files



- Unit of scheduling
  - Scheduler typically limits the length of the pipeline to the number of cores on the node to which the pipeline is assigned for execution
- Input and output files are termed concrete files

Concrete Files

- Used to hold input, output, and executable files
- Single OS file, replicated k times (default k=3)
- Locations and checksums stored in DB

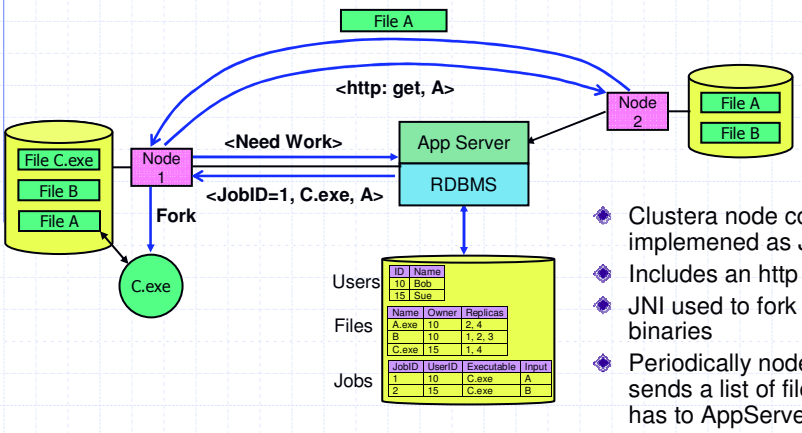


Concrete Job Scheduling

- When idle, node pings server for a job
- Matching is a type of “join” between a set of idle machines and a set of concrete jobs
- Goals include:
  - “Placement aware” scheduling
  - Avoid starvation
  - Job priorities
- Ideal match for a node is one for which both the executable and input files are already present
- Scheduler responds with:

  <jobId, {executable files}, {input files}, {output files}>

Scheduling Example



- Clustera node code is implemened as JVM
- Includes an http server
- JNI used to fork Unix binaries
- Periodically node sends a list of files it has to AppServer

Logical Files and Relational Tables

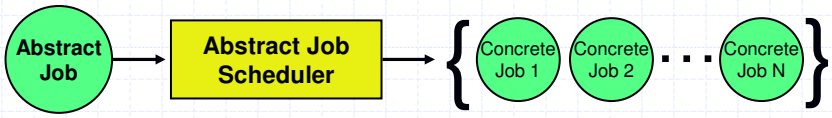
- Logical File
  - Set of one or more concrete files
    - Each concrete file is analogous to a partition of a GFS file
  - Application server automatically distributes the concrete files (and their replicas) on different nodes
  - DB used to keep track of everything
    - File owner, location of replicas, version information, concrete file checksums
- Relational Table
  - Logical File + Schema + Partitioning Scheme
  - Concrete files are treated as separate partitions

Basis of Clustera Extensibility

- Four key mechanisms
  - Concrete Jobs
  - Concrete Files
  - Logical files and relational tables
  - Abstract jobs and abstract job scheduler

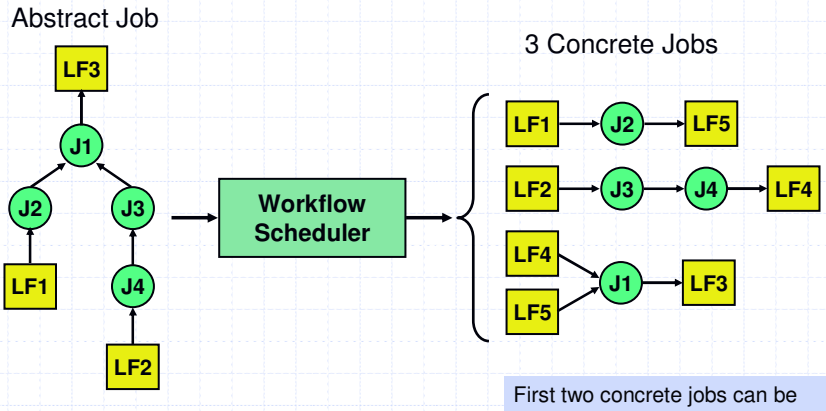
Abstract Job Scheduler

- Sort of a “job compiler”



- Concrete jobs are the unit of scheduling and execution
- Currently 3 types of abstract job schedulers
  - Workflow scheduler
  - Map/Reduce scheduler
  - SQL scheduler

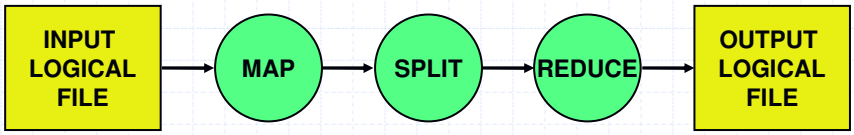
Workflow Scheduler Example



First two concrete jobs can be submitted immediately to the concrete job scheduler. Third must wait until first two have completed.

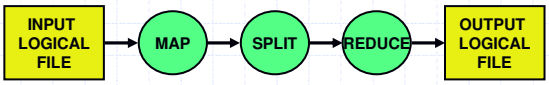
Map Reduce Jobs in Clustera

- Abstract Map Reduce job consists of:
  - Name of logical file to be used as input
  - Map, Split, and Reduce executables
  - Desired number of reduce tasks
  - Name of output logical file

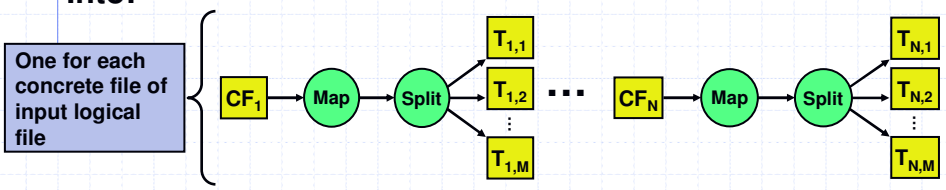


Map Reduce Abstract Scheduler

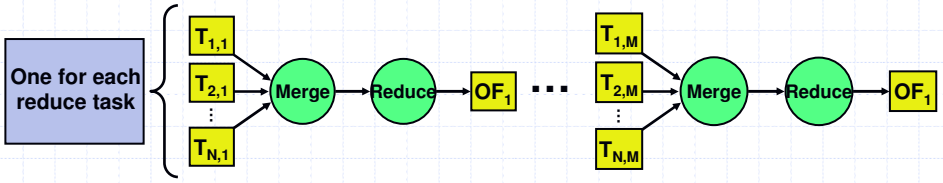
Compiles:



Into:



And:



Clustera SQL

- An abstract SQL specification consists of
  - A set of input tables
  - A SQL query
  - An optional join order
- The Clustera SQL compiler is *not* as sophisticated as a general query optimizer
  - But could be!
- Limitations
  - No support for indices
  - Only equi-joins
  - Select/Project/Join/Aggregate/GroupBy queries only

SQL Example

Tables

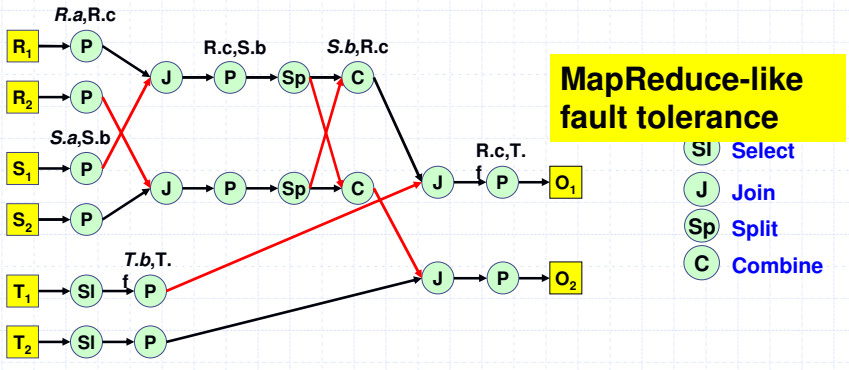
R (*a*, b, c), S (*a*, b, d), T (*b*, e, f) (hash

Files corresponding to red edges are materialized

Query:

Select R.c, T.f from R, S, T where R.a = S.a and S.b = T.b and T.f = X

Concrete job schedule generated (for 2 concrete files per table):

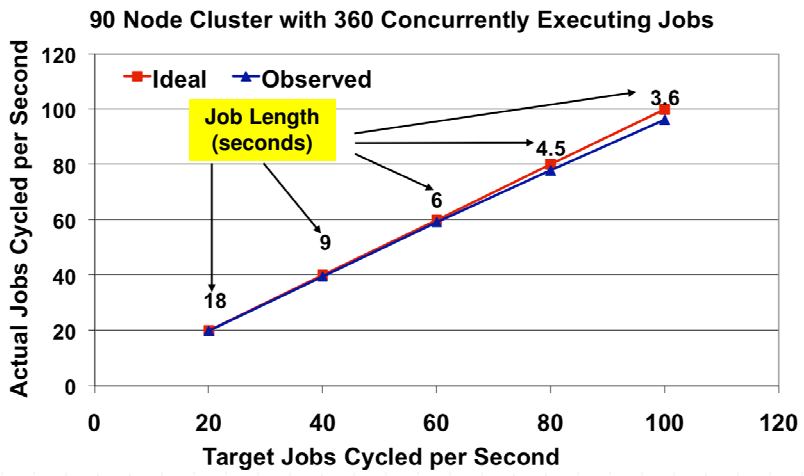


MapReduce-like fault tolerance

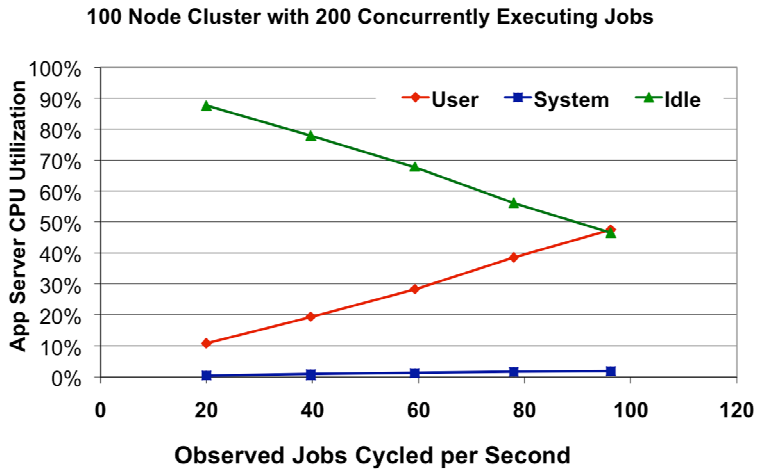
Some Results

- System Configuration
  - 100 node cluster with 2.4Ghz Core 2 Duo CPU, 4GB memory, two 320GB 7200 RPM drives, dual gigabit Ethernet
  - Two Cisco C3560G-48TS switches
    - Connected only by a single gigabit link
  - JBoss 4.2.1 running on 2.4Ghz Core 2 Duo, 2GB memory, Centos 2.6.9
  - DB2 V8.1 running on Quad Xeon with two 3Ghz CPUs and 4GB of memory
  - Hadoop MapReduce Version 0.16.0 (latest version)

Server Throughput

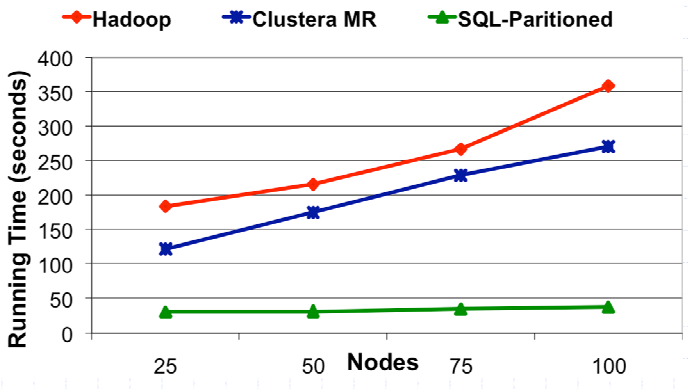


Server Throughput

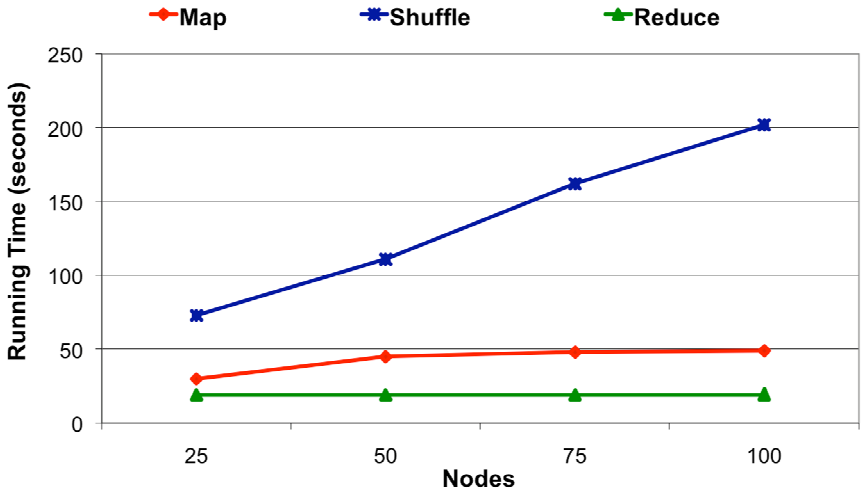


Map-Reduce Scaleup Experiment

- Map Input/Node: 6M row TPC-H LineItem table (795MB)
- Query: Count(*) group by orderKey
- Map Output/Node: 6M rows, 850MB
- Reduce Output/Node: 1.5M rows, 19MB



Clustera MR Details



Why?

- Due to the increase in amount of data transferred between the map and reduce tasks

# of Nodes	Total Data Transferred
25	21.4 GB
50	42.8 GB
75	64.1 GB
100	85.5 GB

SQL Scaleup Test

- SQL Query:

```
SELECT l.okey, o.date, o.shipprio, SUM(l.eprice)
FROM lineitem l, orders o, customer c
WHERE c.mkstsegment = 'AUTOMOBILE' and o.date < '1995-02-03'
    and l.sdate > '1995-02-03' and o.ckey = c.ckey and l.okey = o.okey
GROUP BY l.okey, o.date, o.shipprio
```

- Table sizes
  - Customer: 25 MB/node
  - Orders: 169 MB/node
  - LineItem: 758 MB/Node
- Clustera SQL Abstract Scheduler
- Hadoop + Datajoin contrib package

Partitioning Details

Query GroupBy [(Select (Customer)) Join (Select (Orders)) Join LineItem]

Hash Partitioned Test:

Customers & Orders hash partitioned on ckey
LineItem hash partitioned on okey

Round-Robin Partitioned Test:

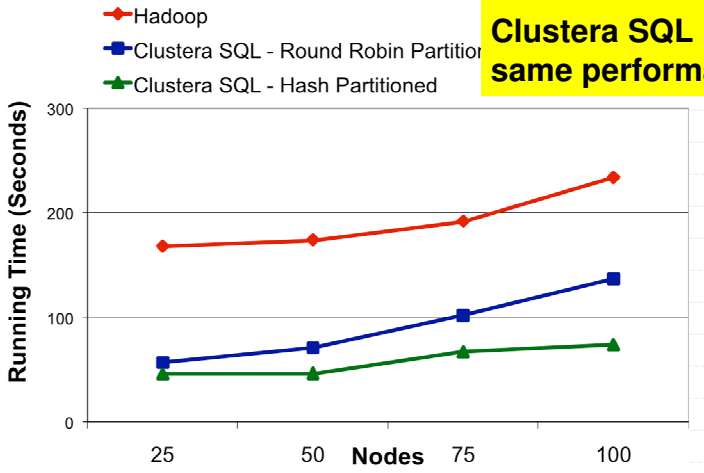
Tables loaded using round-robin partitioning
Workflow requires 4 repartitions

# Of Nodes	Total Data Shuffled (MB)	
	Hash Partitioned Tables	Round-Robin Partitioned Tables
25	77	2122
50	154	4326
75	239	6537
100	316	8757

SQL Scaleup Results

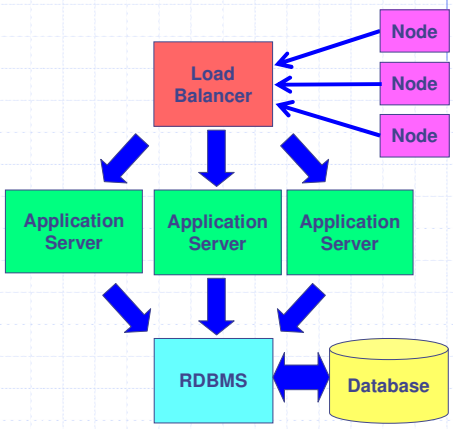
At 100 nodes, 1000s of jobs and 10s of 1000s of files

Clustera SQL has about same performance DB2



Application Server Evaluation

- Clustera design predicated on the use of clustered app servers for
  - Scalability
  - Fault Tolerance
- When clustered, must select a caching policy
  - With no caching, processing is exactly the same as non-clustered case
  - With caching, app servers must also coordinate cache coherence at xact commit

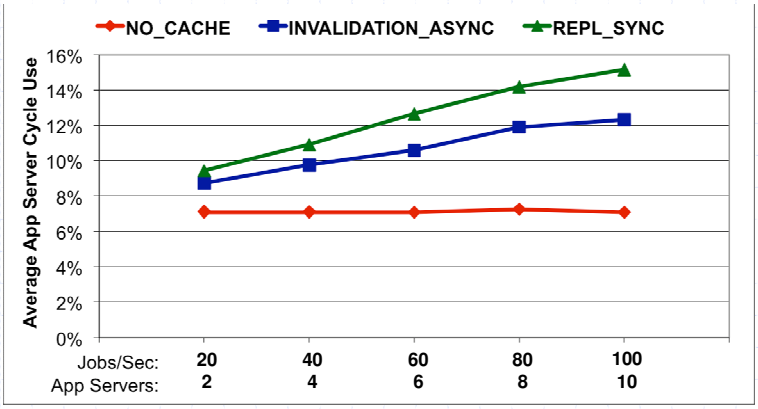


Experimental Setup

- 90 nodes running 4 single-job pipelines concurrently
  - 360 concurrently running jobs cluster-wide
- Load Balancer (Apache mod_jk)
  - 2.4 GHz Intel Core2 Duo, 2GB RAM
- Application Servers (JBoss 4.2.1, TreeCache 1.4.1)
  - 1 to 10 identical 2.4 GHz Intel Core2 Duo, 4GB RAM, no cache limit
- DBMS (IBM DB2 v8.1)
  - 3.0 GHz Xeon (x2) with HT, 4GB RAM, 1GB buffer pool
- Job queue preloaded with fixed-length "sleep" jobs
  - Enables targeting specific throughput rates

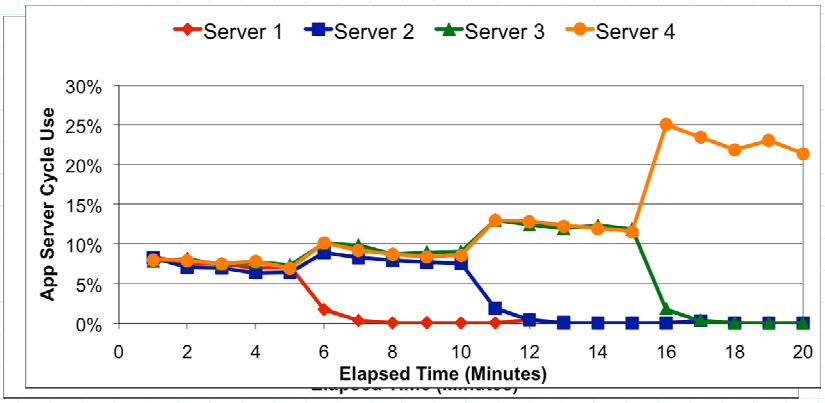
Evaluation of Alternative Caching Policies

- Caching alternatives:
  no caching, asynchronous invalidation, synchronous replication
- 90 Nodes, 4 concurrent jobs/node



Application Server Fault Tolerance

- Approach: maintain a target throughput rate of 40 jobs/sec; start with 4 servers and kill one off every 5 minutes; monitor job completion, error rates
- Key insight: Clustera displays consistent performance with rapid failover – of 47,535 jobs that successfully completed, only 21 had to be restarted due to error



Application Server Summary

- Clustera can make efficient use of additional application server capacity
- The Clustera mid-tier "scales-out" effectively
  - About same as "scale-up" – not shown
- System exhibits consistent performance and rapid failover in the face of application server failure
- Still two single points of failure. Would the behavior change if we:
  - Used redundancy or round-robin DNS to set up a highly available load balancer?
  - Used replication to set up a highly available DBMS?

Summary & Future Work

- Cluster management is truly a data management task
- The combination of a RDMS and AppServer seems to work very well
- Looks feasible to build a cluster management system to handle a variety of different workload types
- Unsolved challenges:
  - Scalability of really short jobs (1 second) with the PULL model
  - Make it possible for mortals to write abstract schedulers
- Bizarre feeling to walk away from a project in the middle of it