Fermilab

Comparison of the Frontier Distributed Database Caching System with NoSQL Databases

CHEP 2012

Dave Dykstra
dwd@fnal.gov

Fermilab is operated by the Fermi Research Alliance, LLC
under contract No. DE-AC02-07CH11359 with the United States Department of Energy

Outline

Goal: increase familiarity with Frontier & NoSQL

- Common characteristics of NoSQL databases
- The Slashdot Effect
- Frontier Distributed Database Caching system characteristics
- CMS Frontier/Squid deployment examples
- Comparison of Frontier to NoSQL in general
- Comparisons of specific NoSQL systems
- Conclusions

NoSQL common characteristics

- “NoSQL” denotes a large variety of Database Management Systems (DBMS)
- Primary unifying characteristic: not a Relational Database Management System (RDBMS)
  - Generally nested key/value instead of row/column
    - Run-time flexibility, doesn't need pre-defined schemas
  - Most don't support the RDBMS standard Structured Query Language SQL
- Most popular NoSQL DBs support being distributed and fault-tolerant – highly scalable on commodity HW
- Most give up atomicity of updates (ACID) and instead have eventual consistency (BASE)

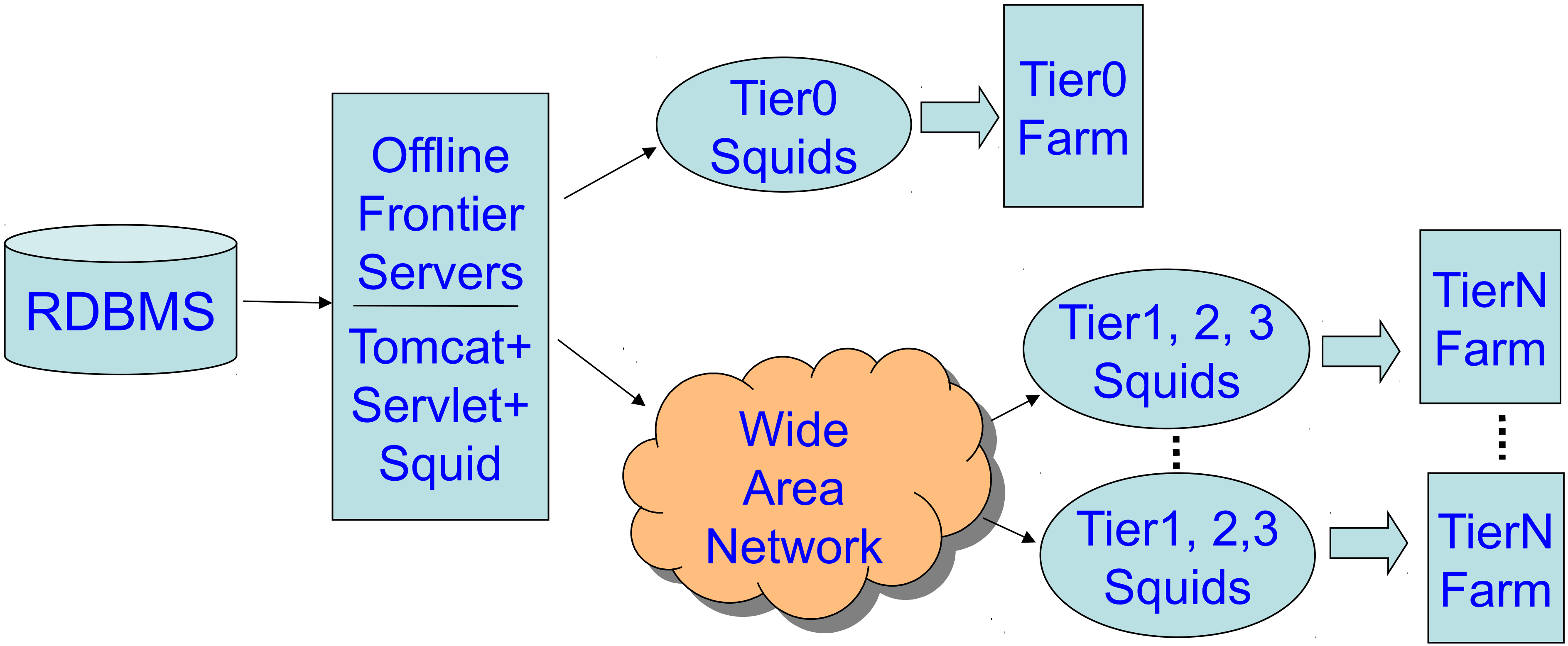
The Slashdot Effect

- Slashdot Effect (or, slashdotting): when a too-small server is overwhelmed by the same request from too many clients
  - Named for slashdot.org, a very popular technical news aggregator website that often hyperlinks to less-popular sites
- For web servers, usual solution is to use a Content Delivery Network (CDN) that either replicates or caches the objects around the world
- Some database applications have similar need

Frontier characteristics

- The Frontier Distributed Database Caching System is designed for the Slashdot Effect – many readers of same data, few writers
  - Distributes RDBMS SQL queries (not “NoSQL”)
  - RESTful, so cacheable with standard web proxy caches (we use Squid)
  - Web caches on client premises make ideal CDN
    - Most network traffic on LAN, scalable as needed
    - Practically maintenance-free
  - Simultaneous same requests collapsed to one
  - Simultaneous different requests queued in Frontier server

CMS Offline Frontier/Squid Conditions deployment

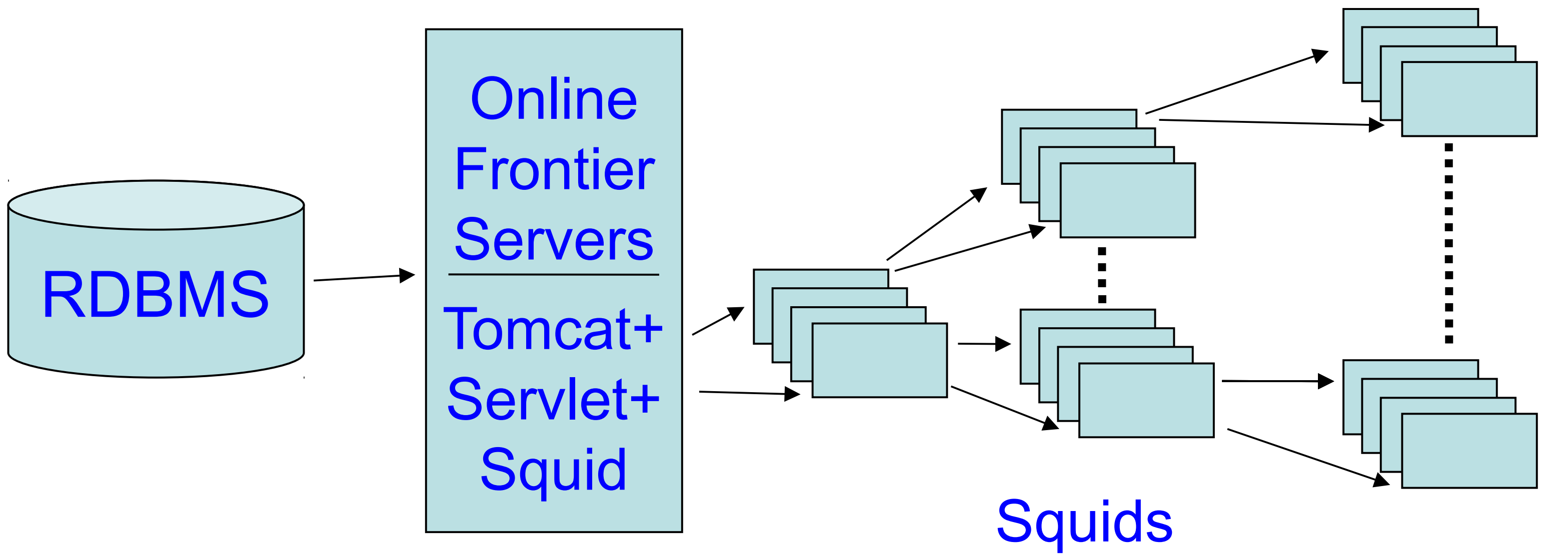


- Only custom software is Frontier servlet in Tomcat and frontier_client in application on worker node farms
- Planning to replicate RDBMS & Frontier servers for availability

CMS Offline Frontier/Squid Conditions stats

- For Tier 0, 1, & 2 (not counting Tier 3):
  - Average 500,000 total Frontier requests per minute, aggregate average total 500MB/s
    - Bursts at sites are much higher than average
- The 3 central server Squids at CERN only get 4,000 average requests per minute, 0.5MB/s
  - Factor of 125 improvement on requests and 1000 on bandwidth (not counting Tier 3)
    - Difference primarily because of If-Modified-Since
- Vast majority of jobs read very quickly because results are already cached & valid in local Squids

CMS Online Frontier/Squid Conditions deployment



- Squid placement is very flexible for more bandwidth
  - Hierarchy of Squids on every worker node
  - Blasts data to all 1400 nodes in parallel

Squid & Frontier limits

- Frontier Tomcat servlet
  - 3-year old 8-core machine (Xeon L5420 @ 2.5Ghz):
    - Without compression, easily saturates 1Gbit network out
    - With gzip compression, drops to 25MB/s out (but saves much bandwidth later in the caches)
    - Adds 1/3rd overhead after gzip to avoid binary data
- Squid
  - 2-year old machine (Xeon E5430 @ 2.66Ghz):
    - Saturates 2Gbit network with one single-thread Squid
  - modern machine (Opteron 6140):
    - Up to 7Gbps on 10Gbit network with a single-thread Squid
    - Can get full throughput with two Squid2s on same port

Frontier vs. NoSQL in general

	Frontier	NoSQL in general
DB structure	Row/column	Nested key/value
Consistency	ACID DB, eventual reads	Eventual
Write model	Central writing	Distributed writing
Read model	Many readers same data	Read many different data
Data model	Central data, cache on demand	Distributed data, copies
Distributed elements	General purpose	Special purpose

Specific NoSQL systems

- Systems currently used in production by CMS or ATLAS
  - MongoDB
  - CouchDB
  - Hadoop HBase
  - Cassandra

MongoDB

{name: "mongo", type:"DB"}

- "Mongo" for "humongous" - for big, cheap data
- Stores binary JSON (JavaScript Object Notation) data
- More similar features to RDBMS than most NoSQL
  - Any field can be memory-indexed for performance
  - Flexible queries
    - By fields, ranges, and regular expressions
- Only one write server per data item
  - Copies are read-only, can take over as master if master goes down
- Scales by sharding, splitting writing of different data to different servers
  - Not great at Slashdot effect

MongoDB cont'd

- Used by CMS for Data Aggregation Service (DAS)
  - Needed the dynamic structure, liked other features
  - Not a big installation though, only one server
  - See poster 184 Thursday
- Supports MapReduce for distributing query processing to where the data is
  - An ATLAS evaluation showed this didn't work well but it is supposed to be better now in version 2.0
    - More in talk after this one

CouchDB

- Stores JSON
- RESTful interface
  - Can use http proxy caches where needed
  - Also easy to insert authentication proxy
- Automated, low-maintenance replication
- All copies get all data, all can read and write
- Uses MultiVersion Concurrency Control (MVCC)
  - Feature of RDBMS – transactions, ACID
  - Readers get a consistent view
  - Writing doesn't block reading
    - Write conflicts have to be resolved by application, however

CouchDB cont'd

- Querying is done by creating “views” defined by JavaScript functions
  - Uses MapReduce paradigm for the functions but processing is not distributed among multiple servers
- Used by CMS for several Workload Management functions
  - CouchDB data replicated between CERN and Fermilab, 3 replicas at CERN and 4 at Fermilab
  - Again, see poster 184 on Thursday

Hadoop HBase

- HBase is built on Hadoop Distributed FileSystem
  - HDFS automatically distributes files and replicates them across a cluster
  - Tunable replication level
  - Very reliable and automated for large amount of data
  - Good for big installations, not small
- Modeled after Google's BigTable
  - Billions of rows with millions of columns
  - Good for search engine-like applications
- Very good at distributed MapReduce

HBase cont'd

- Also has SQL and RESTful API add-ons
- Used by ATLAS distributed data manager for log analysis and accounting on a 12-node cluster
  - Original straightforward accounting summary method was 8 to 20 times faster than same method on a shared Oracle, depending on replication level
  - More in talk after this & poster 425 Thursday
- HBase was recognized by the WLCG Database Technical Evolution Group as having greatest potential impact of all the NoSQL technologies
- CERN IT is setting up a cluster

Cassandra

- Like HBase, modeled after Google BigTable
- All nodes are masters, decentralized control for geographically distributed fault tolerance
  - Dynamic re-configuration with no downtime
- Keys and values can be any arbitrary data
- Has static "column families" used like indexes in RDBMS
- Tunable consistency from always consistent to eventual consistency
- Tunable replication level and in-memory caching

Cassandra cont'd

- Can do MapReduce via Hadoop add-on
- Originally written by Facebook, but they abandoned it in favor of HBase
- Used in production by ATLAS PanDA monitoring system
  - Hosted on 3 high-power nodes at BNL, 24 cores each, 1TB of RAID0 SSDs each
  - See poster 359 on Tuesday

Comparison summary

	MongoDB	CouchDB	HBase	Cassandra	Frontier
Stored data format	JSON	JSON	Arbitrary	Arbitrary	SQL types
Flexible queries	Yes	No	No	No	Yes
Distributed write	No	Yes	No	Yes	No
Handles Slashdot Effect	No	Yes, best w/squid	If scaled sufficiently	If scaled sufficiently	Yes
Does well with many reads of different data	Yes	Yes	Yes	Yes	No
RESTful interface	No	Yes	Add-on	No	Yes
Consistency	Eventual	ACID DB, eventual read	Mixed	Tunable	ACID DB, eventual read
Distributed MapReduce	No	No	Yes	Add-on	No
Replication	Few copies	Everything	Tunable	Tunable	Caching

Conclusions

- NoSQL databases have a wide variety of characteristics, including scalability
- Frontier+Squid easily & efficiently adds some of the same scalability to relational databases when there are many readers of the same data
  - Also enables clients to be geographically distant
- CouchDB with REST can have same scalability
- Hadoop HBase has most potential for big apps
- There are good applications in HEP for many different Database Management Systems