A Key Value Store that Supports Strict SLAs and the Applications that Need it

Christopher Stewart
The Ohio State University
You May Remember Me From Such Research As...

**Performance Models for Services**
- Low-level software to collect data
- Apply queuing models in practice
- Machine learned sub models

**Eurosys 07, ASPLOS 08, ATC 08, SPAA 08**

**Performance Anomaly**
- Formal defn' of anomaly
- Root cause diagnosis
- Avoid by learning when

**Sigmetrics 09, HotDep 06, LADiS 08, IISWC 08**

**Power Del.**

**SLA Management**

**Some Joules are More Precious Than Others**
- HotPower 09, WEED 11, NOMS 12

---

**Papers in the pipeline**
- My favorite work thus far
- Feedback appreciated

---

*Note: The timeline shows the progression of research topics from 2003 to 2010.*
Current Research Goal

- Process data analytics on fast storage
- Use many shared-nothing storage nodes
- Many machine-to-machine interactions before human

**Systems support for very low latency distributed services**

*Every storage access completes in milliseconds*
Outline

1. **The Demand for Very Strict, Low-Latency SLAs**
2. A Key Value Store for Very Strict, Low-Latency SLAs
3. Results
4. Conclusion
Emerging Workloads & Very Strict SLAs

Service Level Agreement (SLA)

- 95% of storage accesses will complete in 1 sec.
- **Low latency → millis**
- **Very strict → 99.9%**

<table>
<thead>
<tr>
<th>Traditional Services</th>
<th>Emerging Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load balancer</td>
<td>Web server</td>
</tr>
<tr>
<td>Web server</td>
<td>Middleware</td>
</tr>
<tr>
<td>Application server</td>
<td>Data store</td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td>Zookeeper</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>BigTable</td>
</tr>
<tr>
<td></td>
<td>Dynamo</td>
</tr>
</tbody>
</table>

1. Scientific Cloud
2. Data Analytics
3. Managing SLAs
Emerging Workloads & Very Strict SLAs

- Emerging services issue a lot of accesses for each user request
  - Simple semantics
  - More complex service
- Emerging services have more dependencies

1. Scientific Cloud
2. Data Analytics
3. Managing SLAs

Traditional Services

- Load balancer
- Web server
- Application server
- Database
  - MySQL
  - IBM DB2

Emerging Services

- Web server
- Middleware
  - transactions
  - bus. logic
  - SQL wraps
- Data store
  - Zookeeper
  - BigTable
  - Dynamo
  - Cassandra
Some scientific simulations can use the Cloud

- Non Goal: Replace the super computer
- Medium size simulations use only a portion of the cores in a super computer
- Loosely coupled agents
- Scale response time from hours to minutes

Cloud-based Smart Grid Simulation

Emerging Workloads & Very Strict SLAs

1. Scientific Cloud
2. Data Analytics
3. Managing SLAs
Emerging Workloads & Very Strict SLAs

- **Example: Smart-Grid Simulation (this talk)**
  - Operators adjust production for wind surges, hot days
  - # of simulations depend on conditions (elastic)
  - Simulations of smart grid help (if done in 5 min.)
  - Each simulation >10K storage accesses

- Other examples: Hurricane sim., NAS Bench. [HPDC-12]
Emerging Workloads & Very Strict SLAs

- Advertising remains a driving force on the web
  - Extract phrases from document (e.g., email), Feed phrases into keyword predictor that outputs $p(\text{phrase is a keyword})$, ads use weighted payout

- Targeting ads based on user input increases profits
  - Recent searches at a search engine [Yih-WWW-2002] can improve keyword selection by 25—50%
  - Crowds and deep context can help also

- Systems challenge:
  - Data changes too frequently to cache
  - The more up-to-date the analysis the better
  - **Extract a lot of features for each query**
Data analytics is akin to Deep Q & A

- Identify (dynamic) features in a document, build predictor on the fly [compare extracted features to every item in a dataset], provide confident answer
- IBM Watson, must respond in seconds
- Number of storage accesses depends dataset size

He said, “Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it.”
Emerging Workloads &
Very Strict SLAs

- Deep Q & A and data sets
  - TREC 9 articles (used by the Watson team)
  - Crawled USENIX and ACM conference pages
    - Who was PC chair of ...?
  - *Emails from a renewable-powered IMAP cache*

![Diagram showing Map-Reduce Leader, Distributed Storage, Storage System: Cassandra, HDFS, Dynamo, etc. with Watson or (in our case) Open Ephyra]
Emerging Workloads & Very Strict SLAs

■ Traditional view: SLA reflects common response times; set conservatively; improves over time

■ Emerging view: SLA reflects human tolerance; independent of technology; holds steady over time

■ Meet SLA using as few resources as possible, then exploit competitive advantages

  Lower costs
  • Route requests to cheap energy
  • Route requests to avoid carbon penalties

  Increase revenue
  • Strategically invest carbon offsets to attract new, green users (i.e., think Prius---Going Green To Be Seen)
Emerging Workloads & Very Strict SLAs

Greenmail
- IMAP Cache
- Browser based
- Hosted on GreenQloud, 100% renewable powered
- Will provide differentiated SLA and differentiated carbon footprint
- Data analytics?
Emerging Workloads & Very Strict SLAs

1. Scientific Cloud
2. Data Analytics
3. Managing SLAs

- Deep Q & A under response time bounds (also data analytics)
- Zoolander: A Key Value Store That Meets Very Strict, Low Latency SLAs
- SLA Management (adaptive green hosting)
Outline

1. The Demand for Very Strict, Low-Latency SLAs

2. **A Key Value Store for Very Strict, Low-Latency SLAs**

3. Results

4. Conclusion
Replication and partitioning are the most widely used approaches to manage performance.

- Improve throughput by adding nodes
- Achieve low latency via fewer accesses per node
  - Hot spots and consistency are common challenges

1. Rep. for Predictability
2. Performance Model
3. Model Evaluation
Problem: Even under small partitions and light loads, most key value stores can't meet very strict SLAs

- Access a small key range, no concurrent accesses

Consistent across:
- Cassandra (Facebook)
- Zookeeper (Yahoo!)
- Memcached (Youtube)
One reason for heavy tails: Background jobs

- Cassandra by default buffers writes
- A buffer flush every second (if there was a write)
- With many writes, buffer flush takes ~10ms
- A typical access takes <1ms
- 5X expected slowdown if stuck behind the buffer flush, but only 5% of accesses affected
- Also, garbage collection, DNS timeouts, data copy

Properties

- Not correlated with any specific access
- Really big delays
Heavy tails have a big impact

- Smart-grid simulation with long delay injected into 2% of accesses versus small delay into 100%
- Can't mask long delays by overlapping computation

---

Zoolander

1. Rep. for Predictability
2. Performance Model
3. Model Evaluation

---

Slide 19
**Key Insight:** Revisit replication for predictability, an old, dumb idea whose time has come [mogul, hotos]

- Old, simple approach: Make N copies, send all accesses to every copy, take the first response
- Doesn't improve throughput; dismissed (dumb)
- Does reduce variance, exactly what we need now!
Other's have used replication for predictability also

- SCADS Manager, FAST 2010
  SCADS Manager ensured high SLA by migrating overloaded partitions and by using the fastest read from two nodes

- Mantri, OSDI 2010
  Mantri meets tight map-reduce deadlines by running two copies of certain nested tasks

We provide full support for this approach

- Scale the number of duplicates (i.e., more than 2)
- Understand its effect on performance
- Deal with writes
Zoolander, our key value store, supports:

- Read and write access
- Supports scaling out (i.e., adding nodes)
- **Decides when to scale this approach (model)**
- Supports a wide range of consistency models
- Adaptation to workload changes, Failure, recovery, integration with existing key value stores, ...,

Used throughout my research group now and could face real, user workloads beginning summer
Partitioning and traditional replication have been shown to help with meeting SLAs

System managers dilemma:

- An SLA violation just happened
  - 4% of requests took longer than 15ms to complete
- I provisioned an extra EC2 instance (scale out)
  - Should I partition the keys on the most heavily used node or should I replicate a node for predictability?

Our performance model outputs the expected service level after applying each approach
Performance Model
(First principles)
(Geometric series)
(Queuing theory)

arrival rate
service time
distributions
network latency

Target SLA
service level
latency bound

iterate
replication policy
(aka data layout)

Expected Service Level

4 1-duplicate partitions

2 2-duplicate partitions

A B C D

Zoolander

1. Rep. for Predictability
2. Performance Model
3. Model Evaluation
Partitioning and tradition replication are widely used and well supported

- Key simplification: Bias for those approaches
- Ignore hotspots, consistency overhead
- Assume even distribution of accesses

Challenge: We need an accurate model for replication for predictability; Is it really useful?
Zoolander

1. Rep. for Predictability
2. Performance Model
3. Model Evaluation

- Treat each access independently
  - What is the probability of an SLA violation?
  - Use CDF
  - $1 - \Phi(\tau)$

- Consider dependencies
  - Queuing delay
  - Network latency
  - Workload changes
(1) Processing time

- Initially, \( N = 1, \hat{s}_1 = \Phi_0(\tau); \)
- Then, \( N = 2, \hat{s}_2 = (1 - \Phi_0(\tau)) \Phi_1(\tau) + \Phi_0(\tau) = (1 - \hat{s}_1) \Phi_1(\tau) + \hat{s}_1 \)
- \( N = 3, \hat{s}_3 = (1 - \hat{s}_2) \Phi_2(\tau) + \hat{s}_2 = \)
  ...
- \( \hat{s}_n = (1 - \hat{s}_{n-1}) \Phi_{n-1}(\tau) + \hat{s}_{n-1} \)
- Recursively, we can get

\[
\hat{s} = \sum_{n=0}^{N-1} \left[ \Phi_n(\tau) \prod_{i=0}^{n-1} (1 - \Phi_i(\tau)) \right]
\]

\[
\hat{s} = 1 - (1 - \Phi(\tau))^N \quad \text{If all CDFs are same, its a geometric series}
\]
Consider dependencies

- Queuing delay
- Network latency

Insight: These add to latency, making the latency bound lower

\[ \tau_n = \tau - \mu_{net} - \mu_{queue} \]

- Finally, we get

\[ \hat{s} = \sum_{n=0}^{N-1} [\Phi_n(\tau_n) \times \prod_{i=0}^{n-1} (1 - \Phi_i(\tau_i))] \]

Workload changes need additional support, i.e., runtime monitoring
Micro-benchmark: Issue 100% writes
No concurrency, network latency 150 micro sec.
Run 1M accesses, add a duplicate after each
ZK = 1, Absolute percentage-point error
Caveat:

- Does not consider hot spots/consistency
- Percentage points are not the same as latency
- Latency bound error was always below 10%
- Results hold across reads, ZK=3, and hardware
What about queuing? Exactly.

Key point #1: Rep for Pred. achieves service levels the other approaches can't (even with model bias)

Key point #2: Rep for Pred. scales to moderate util. as tau increases

Key point #3: A mixed approach provide the best of both worlds
Outline

1. The Demand for Very Strict, Low-Latency SLAs
2. A Key Value Store for Very Strict, Low-Latency SLAs
3. Results
4. Conclusion
Push all state changing writes through a multicast repeater (ideally hardware)

Allows: SC, Read-my-own, and Write Order

Swap out key value stores on the back end.
Support Zookeeper & Cassandra

Results

1. Gridlab-D
2. Open workload
3. End-to-End Tests

Swap out key value stores on the back end.
Support Zookeeper & Cassandra

Push all state changing writes through a multicast repeater (ideally hardware)

Allows: SC, Read-my-own, and Write Order

Low overhead, callback

Systems data monitoring

Results

1. Gridlab-D
2. Open workload
3. End-to-End Tests

Swap out key value stores on the back end.
Support Zookeeper & Cassandra

Push all state changing writes through a multicast repeater (ideally hardware)

Allows: SC, Read-my-own, and Write Order

Low overhead, callback

Systems data monitoring
Set up Gridlab-D, a smart grid simulator at PNNL

At every time step, uses agent-based simulation to compute electricity usage

Stored global variables, input files, and output files in Zoolander
Results

- Consolidate simulations on as few nodes as possible
- Still care about the response time of each
  - 10ms latency bound
  - Recall rep for pred. doesn't improve unconstrained throughput
- Mixed approach doubles 'goodput'

1. Gridlab-D
2. Open workload
3. End-to-End Tests

![Graph showing concurrency and nodes needed](image)
Zoolander improves end-to-end execution time ~20%

- Compared equal sized cluster (8X compared to 1 node)
- It achieves this by reducing 99\textsuperscript{th} percentile by 68%
- Gains scale to 32 EC2 nodes
Outline

1. The Demand for Very Strict, Low-Latency SLAs
2. A Key Value Store for Very Strict, Low-Latency SLAs
3. Results
4. Conclusion
Emerging workloads need storage systems that can ensure low latency on thousands of accesses.

We revisit *replication for predictability*, a scale-out approach that improves latency for slow accesses when traditional approaches can't.

Zoolander, our key value store, fully supports this approach, including: read/write, performance model, diverse consistency levels, failure recovery, ....

Smart-grid simulations (Gridlab-d) saw 20% end-to-end speedup using Zoolander instead of an equal-sized system based on only traditional approaches.
Conclusion

Scientific Computing on the Cloud

Zoolander: A Key Value Store That Supports Very Strict, Low Latency SLAs

- Daiyi Yang, StumbleUpon.com VC finalist
- Aniket Chakrabarti, 1st year PhD
- Alex Bunch, BS w/ Honors CMU 2012

Deep Q & A under response time bounds (also data analytics)

- Jaimie Kelley, 2nd year PhD
- Nan Deng, 3rd year PhD ISSST Award paper
- Nan Deng, 3rd year PhD ISSST Award paper

SLA Management (Adaptive Green Hosting)

- Nan Deng, 3rd year PhD ISSST Award paper
- Alex Bunch, BS w/ Honors CMU 2012
- Daiyi Yang, StumbleUpon.com VC finalist
- Aniket Chakrabarti, 1st year PhD

Conclusion