Formal Modeling and Analysis of Google's Megastore in Real-Time Maude

Jon Grov and Peter Ölveczky

University of Oslo

In honor of Kokichi Futatsugi

Thanks to Indranil Gupta (UIUC) and UIUC’s Center for Assured Cloud Computing (support by AFOSR Grant FA8750-11-2-0084)
CLOUD COMPUTING  (FIGURE FROM http://www.smartgridnews.com/)
Computing as a Utility
Cloud Computing

Multi-tenant solution provided by vendor

Automated backups, uptime, SLA, maintenance

Automated upgrades

Elastic, pay as you go - scale up or down

Modern web based integration

Web and mobile - access from anywhere

(Figure from http://osarena.net/faqs/tosese-cloud-ipiresies-pos-na-tis-organoso.html)
Data in the Cloud

- Availability and scalability: data must be replicated
**Data in the Cloud**

- **Availability and scalability**: data must be **replicated**

“Eventual consistency” OK for some applications
Eventual Consistency

“Eventual consistency” OK for some applications
Eventual Consistency

“Eventual consistency” OK for some applications
**Eventual Consistency**

“Eventual consistency” OK for some applications
“Eventual consistency” OK for some applications

... but not others:

- banking
- stock exchange
- electronic commerce
  - online auctions (eBay)
  - plane tickets ...
- medical (information) systems
Transactions

Databases traditionally provide ACID transactions

- atomicity
- consistency
- isolation (“serializability”)
- durability
Databases traditionally provide **ACID transactions**

- atomicity
- consistency
- isolation ("serializability")
- durability

"It is possible to build your own [transaction support] on any of these systems, given enough additional code. However, the task is so difficult, we wouldn’t wish it on our worst enemy. If you need [transaction support], you want to use a DBMS that provides them; it is much easier to deal with this at the DBMS level than at the application level." (M. Stonebraker)
Megastore:

- Google’s wide-area replicated data store
- Key part of Google’s cloud infrastructure
Megastore: Why Interesting?

- “Widely deployed in Google for several years. Used on more than 100 production applications. Handles more than 3 billion write and 20 billion read transactions daily. Stores nearly a petabyte of primary data across many global datacenters. Available on GAE since Jan 2011.” [Baker et al.]
Megastore: Why Interesting?

• “Widely deployed in Google for several years. Used on more than 100 production applications. Handles more than 3 billion write and 20 billion read transactions daily. Stores nearly a petabyte of primary data across many global datacenters. Available on GAE since Jan 2011.” [Baker et al.]

• Adds (limited) transactions to wide-area replicated data stores
Megastore: Key Ideas (I)

(Figure from http://cse708.blogspot.jp/2011/03/megastore-providing-scalable-highly.html)
Megastore: Key Ideas (I)

- Data divided into **entity groups**
  - Peter’s email
  - Books on rewriting logic
  - Jon’s documents

(Figure from http://cse708.blogspot.jp/2011/03/megastore-providing-scalable-highly.html)
Megastore: Key Ideas (I)

- Data divided into **entity groups**
  - Peter’s email
  - Books on rewriting logic
  - Jon’s documents
- **Replicated transaction log** for each entity group

(Figure from http://cse708.blogspot.jp/2011/03/megastore-providing-scalable-highly.html)
Megastore: Key Ideas (II)

- Node suggests next log entry for an entity group
- **Paxos**: agree on next log entry if concurrent updates

**Before coordination**

- **Site A**: Data, Replicated log
- **Site B**: Data, Replicated log
- **Site C**: Data, Replicated log

**After (successful) coordination**

- **Site A**: Data, Replicated log
- **Site B**: Data, Replicated log
- **Site C**: Data, Replicated log
Megastore: Key Ideas (III)

- Consistency for transactions accessing a single entity group
  - no guarantee if transaction reads multiple entity groups
Megastore: Key Ideas (III)

- **Consistency** for transactions accessing a single entity group
  - no guarantee if transaction reads multiple entity groups
- **ElasTraS, Spinnaker, Calvin, and Microsoft’s Azure:** consistency within each data partition
Summary (I)

Figure 1: Scalable Replication

(figure from J. Baker et al., “Megastore: Providing Scalable, Highly Available Storage for Interactive Services”)
What About the CAP Theorem?

- Consistency + wide-area replication??
What About the CAP Theorem?

• Consistency + wide-area replication??
• Component’s coordinator knows whether data might be inconsistent
Our Work: Motivation

- 12 page informal overview paper of really complex system
- Understand system and guarantees
- Basis for further research and extensions
Our Work

- [Developed and] formalized [our version of the] Megastore [approach] in Real-Time Maude
Our Work

• [Developed and] formalized [our version of the] Megastore [approach] in Real-Time Maude
  • first (public) formalization/detailed description of Megastore
  • “efficient” model of multicast with nondeterministic delays
Our Work

- [Developed and] formalized [our version of the] Megastore [approach] in Real-Time Maude
  - first (public) formalization/detailed description of Megastore
  - “efficient” model of multicast with nondeterministic delays
- 56 rewrite rules (37 for fault tolerance features)
Our Work

• [Developed and] formalized [our version of the] Megastore [approach] in Real-Time Maude
  • first (public) formalization/detailed description of Megastore
  • “efficient” model of multicast with nondeterministic delays
• 56 rewrite rules (37 for fault tolerance features)
• Real-Time Maude simulation and model checking used extensively throughout development of (our) Megastore
A Rewrite Rule (in Failure-Free Setting)

When all operations in the operations list are completed (reads) or buffered (writes), the transaction is ready to commit. All buffered updates are merged into a candidate log entry. If the transaction updates entities from several entity groups, one log entry is created for each group. For each such entity group, the first step is to send the candidate log entry to the leader for the next log position, which was selected during the previous coordination round. The rule for initiating Paxos is modeled as follows:

\[
\text{crl} \ [\text{initiateCommit}] : \\
< \text{SID}' : \text{Site} | \\
\quad \text{entityGroups} \ EGROUPS, \\
\quad \text{localTransactions} : \text{LOCALTRANS} \\
\quad \quad < \text{TID} : \text{Transaction} | \text{operations} : \text{emptyOpList}, \\
\quad \quad \quad \text{writes} : \text{WRITEOPS}, \text{status} : \text{idle} \\
\quad \quad \quad \text{readState} : \text{RSTATE}, \text{paxosState} : \text{PSTATE} > > \\
\]

\[
=> \\
< \text{SID} : \text{Site} | \\
\quad \text{localTransactions} : \text{LOCALTRANS} \\
\quad < \text{TID} : \text{Transaction} | \text{paxosState} : \text{NEW-PAXOS-STATE}, \\
\quad \quad \text{status} : \text{in-paxos} > > \\
\]

\text{ACC-LEADER-REQ-MSGS}

if \ EIDSET := \text{getEntityGroupIds}(\text{WRITEOPS}) /\ \\
\text{NEW-PAXOS-STATE} := \text{initiatePaxosState}(\text{EIDSET}, \text{TID}, \text{WRITEOPS}, \\
\quad \text{SID}, \text{RSTATE}, \text{EGROUPS}) \\
/\ (\text{createAcceptLeaderMessages}(\text{SID}, \text{NEW-PAXOS-STATE})) => \text{ACC-LEADER-REQ-MSGS}
getEntityGroupIds(WRITEOPS) contains entity groups accessed by operations in WRITEOPS, and NEW-PAXOS-STATE contains one record for each entity group. These records contain the log position that TID requests to update and the candidate log entry \( le \). The operator createAcceptLeaderMessages generates an acceptLeaderReq message to the leader of each entity group containing the transaction id TID and candidate log entry.
Another Rewrite Rule (no Failures)

The following rule [...] where a replicating site receives an acceptAllReq message. The site verifies that it has not already granted an accept for this log position (since messages could be delayed for a long time, it checks both the transaction log and received proposals). If there are no such conflicts, the site responds with an accept message, and stores its accept in proposals for this entity group. The record \((\text{TID}', \text{LP} \ \text{SID} \ \text{OL})\) represents the candidate log entry, containing the transaction identifier \(\text{TID}'\), the log position \(\text{LP}\), the proposed leader site \(\text{SID}\), and the list of update operations \(\text{OL}\).

crl \([\text{rcvAcceptAllReq}]\) :

\[
\text{msg} \ \text{acceptAllReq}(\text{TID}, \ \text{EG}, (\text{TID}', \ \text{LP} \ \text{SID} \ \text{OL}), \ \text{PROPNUM}) \ \text{from} \ \text{SENDER} \ \text{to} \ \text{THIS}
\]
< THIS : Site |
entityGroups : EGROUPS
  < EG : EntityGroup |
  proposals : PROPSET, transactionLog : LEL > >
=>
< THIS : Site |
entityGroups : EGROUPS
  < EG : EntityGroup |
  proposals : accepted(SENDER, (TID', LP SID OL), PROPNUM) ;
  removeProposal(LP, PROPSET) > >
dly(acceptAllRsp(TID, EG, LP, PROPNUM) from THIS to SENDER), T)
if not (containsLPos(LP, LEL) or hasAcceptedForPosition(LP, PROPSET))
  /
\ T ; TS := possibleMessageDelay(THIS, SENDER) .
Formal Analysis

- “Monte-Carlo” simulations for performance estimation
- LTL model checking
  - highly nondeterministic setting (many conflicts)
  - limited scenarios
  - super useful to discover many bugs not found during simulation
Performance Estimation

- Key performance measures:
  - average transaction latency
  - number of committed/aborted transactions
- 2 entity groups
- Randomly generated transactions (rate 2.5 TPS)
- Network delays:

<table>
<thead>
<tr>
<th></th>
<th>30%</th>
<th>30%</th>
<th>30%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>London ↔ Paris</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>London ↔ New York</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Paris ↔ New York</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>100</td>
</tr>
</tbody>
</table>
Simulating for 200 seconds (no failures):

<table>
<thead>
<tr>
<th>City</th>
<th>Avg. latency (ms)</th>
<th>Commits</th>
<th>Aborts</th>
</tr>
</thead>
<tbody>
<tr>
<td>London</td>
<td>122</td>
<td>149</td>
<td>15</td>
</tr>
<tr>
<td>New York</td>
<td>155</td>
<td>132</td>
<td>33</td>
</tr>
<tr>
<td>Paris</td>
<td>119</td>
<td>148</td>
<td>18</td>
</tr>
</tbody>
</table>
Performance Estimation (cont.)

- Simulating for 200 seconds (no failures):

<table>
<thead>
<tr>
<th></th>
<th>Avg. latency (ms)</th>
<th>Commits</th>
<th>Aborts</th>
</tr>
</thead>
<tbody>
<tr>
<td>London</td>
<td>122</td>
<td>149</td>
<td>15</td>
</tr>
<tr>
<td>New York</td>
<td>155</td>
<td>132</td>
<td>33</td>
</tr>
<tr>
<td>Paris</td>
<td>119</td>
<td>148</td>
<td>18</td>
</tr>
</tbody>
</table>

- Site failures:
  - mean-time-to-failure **10 seconds per site**
  - mean-time-to repair **2 seconds**

<table>
<thead>
<tr>
<th></th>
<th>Avg. latency (ms)</th>
<th>Commits</th>
<th>Aborts</th>
</tr>
</thead>
<tbody>
<tr>
<td>London</td>
<td>218</td>
<td>109</td>
<td>38</td>
</tr>
<tr>
<td>New York</td>
<td>336</td>
<td>129</td>
<td>16</td>
</tr>
<tr>
<td>Paris</td>
<td>331</td>
<td>116</td>
<td>21</td>
</tr>
</tbody>
</table>
Model Checking (I)

- Desired properties (for finite number of transactions):
  - all transactions finish execution
  - all replicas of an entity must eventually have same value
  - all logs for entity group must eventually have same entries
  - execution was serializable
Model Checking (I)

- Desired properties (for finite number of transactions):
  - all transactions finish execution
  - all replicas of an entity must eventually have same value
  - all logs for entity group must eventually have same entries
  - execution was serializable

- Serializability tricky property
  - construct serialization graph during execution
  - check that graph has no cycles
Model Checking (II)

- **All replicas same unless coordinator invalidated:**
  
  \[ \text{op entityGroupsEqualOrInvalid} : \rightarrow \text{Prop} \text{ [ctor]} . \]
  
  \[ \text{ceq} \{ < S_1 : \text{Site} | \text{coordinator} : \text{eglpl}(\text{EG}_1, \text{LP}) ; \text{EGLP}, \text{entityGroups} : < \text{EG}_1 : \text{EntityGroup} | \text{entitiesState} : \text{ES}_1 > \text{EGS}_1 > < S_2 : \text{Site} | \text{coordinator} : \text{eglpl}(\text{EG}_1, \text{LP}) ; \text{EGLP}, \text{entityGroups} : < \text{EG}_1 : \text{EntityGroup} | \text{entitiesState} : \text{ES}_2 > \text{EGS}_2 > \text{REST} \} \models \text{entityGroupsEqual} = \text{false} \text{ if } \text{ES}_1 \neq \text{ES}_2 . \]
  
  \[ \text{eq} \{ \text{SYSTEM} \} \models \text{entityGroupsEqualOrInvalid} = \text{true} \text{ [owise]} . \]
Model Checking (II)

- All replicas same unless coordinator invalidated:

  \[
  \text{op entityGroupsEqualOrInvalid : \rightarrow Prop [ctor] .}
  \]

  \[
  \text{ceq \{< S1 : Site | coordinator : eglp(EG1, LP) ; EGLP, entityGroups : }
  \]
  \[
  \text{< EG1 : EntityGroup | entitiesState : ES1 > EGS1 >}
  \]
  \[
  \text{< S2 : Site | coordinator : eglp(EG1, LP) ; EGLP, entityGroups :}
  \]
  \[
  \text{< EG1 : EntityGroup | entitiesState : ES2 > EGS2 >}
  \]
  \[
  \text{REST\} \models entityGroupsEqual = false if ES1 \neq ES2 .}
  \]

  \[
  \text{eq \{SYSTEM\} \models entityGroupsEqualOrInvalid = true [owise] .}
  \]

- Execution serializable:

  \[
  \text{op isSerializable : \rightarrow Prop [ctor] .}
  \]

  \[
  \text{eq \{< th : TransactionHistory | graph : GRAPH > REST\}
  \]
  \[
  \models isSerializable = not hasCycle(GRAPH) .}
  \]
Model Checking (II)

- All replicas same unless coordinator invalidated:
  
  \[ \text{op entityGroupsEqualOrInvalid : } \rightarrow \text{Prop [ctor].} \]
  
  \[ \text{ceq } \{ < \text{S1 : Site | coordinator : eglp(EG1, LP) ; EGLP, entityGroups : } \]
  \[ \text{   < EG1 : EntityGroup | entitiesState : ES1 } \rangle \text{ EGS1 } > \]
  \[ < \text{S2 : Site | coordinator : eglp(EG1, LP) ; EGLP, entityGroups : } \]
  \[ < \text{EG1 : EntityGroup | entitiesState : ES2 } \rangle \text{ EGS2 } > \]
  \[ \text{REST} \} \models \text{entityGroupsEqual } = \text{false if } \text{ES1 } \\text{=}/= \text{ES2}. \]

  \[ \text{eq } \{ \text{SYSTEM} \} \models \text{entityGroupsEqualOrInvalid } = \text{true [owise].} \]

- Execution serializable:
  
  \[ \text{op isSerializable : } \rightarrow \text{Prop [ctor].} \]
  
  \[ \text{eq } \{ < \text{th : TransactionHistory | graph : GRAPH } \rangle \text{ REST}\} \]
  \[ \models \text{isSerializable } = \text{not hasCycle(GRAPH).} \]

- Desired property:
  
  \[ \langle [ ] \rangle (\text{allTransFinished } \land \text{entityGroupsEqualOrInvalid} \]
  \[ \land \text{transLogsEqualOrInvalid } \land \text{isSerializable}) \]
Model Checking (III)

• Without fault tolerance:
  • model checked untimed Maude model
  • covers all possible message delays, transaction start times, etc.
  • 3 sites and 3 transactions
Model Checking (III)

- **Without fault tolerance:**
  - model checked *untimed* Maude model
  - covers all possible message delays, transaction start times, etc.
  - 3 sites and 3 transactions

- **With fault tolerance:** model checked *timed* model

<table>
<thead>
<tr>
<th>Msg. delay</th>
<th>#Trans</th>
<th>Trans. start time</th>
<th>#Fail.</th>
<th>Fail. time</th>
<th>Run (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{20, 100}</td>
<td>4</td>
<td>{19, 80} and {50, 200}</td>
<td>0</td>
<td>-</td>
<td>1367</td>
</tr>
<tr>
<td>{20, 100}</td>
<td>3</td>
<td>{10, 50, 200}</td>
<td>1</td>
<td>60</td>
<td>1164</td>
</tr>
<tr>
<td>{20, 40}</td>
<td>3</td>
<td>20, 30, and {10, 50}</td>
<td>2</td>
<td>{40, 80}</td>
<td>872</td>
</tr>
<tr>
<td>{20, 40}</td>
<td>4</td>
<td>20, 20, 60, and 110</td>
<td>2</td>
<td>70 and {10, 130}</td>
<td>241</td>
</tr>
<tr>
<td>{20, 40}</td>
<td>4</td>
<td>20, 20, 60, and 110</td>
<td>2</td>
<td>{30, 80}</td>
<td>DNF</td>
</tr>
<tr>
<td>{10, 30, 80},and</td>
<td></td>
<td>{30, 60, 120}</td>
<td>3</td>
<td>{30, 80}</td>
<td>DNF</td>
</tr>
<tr>
<td>{10, 30, 80},and</td>
<td></td>
<td>{30, 60, 120}</td>
<td>3</td>
<td>60</td>
<td>DNF</td>
</tr>
</tbody>
</table>

DNF stands for *don’t know*.
Teaser: Megastore-CGC

We have developed Megastore-CGC

• consistency for transactions accessing multiple entity groups
Teaser: Megastore-CGC

We have developed Megastore-CGC

• consistency for transactions accessing multiple entity groups
• performance “same” as for Megastore
  • Megastore sites implicitly order transactions on multiple entity groups
  • “piggy-back” additional information onto Megastore messages
Conclusions

• Developed (own) model of Megastore from limited information
  • first (and only (w/ Cassandra/Maude)?) formal analysis of transactional data stores
Conclusions

• Developed (own) model of Megastore from limited information
  • first (and only (w/ Cassandra/Maude)?) formal analysis of transactional data stores
• Allowed us to develop non-trivial extension of Megastore
  • first replicated data store (design) with ACID transactions accessing multiple “entity groups” (?)
Congratulations, Kokichi, and Welcome Back to Oslo!