Modeling, Analyzing, and Extending Megastore using Real-Time Maude

Jon Grov\textsuperscript{1} and Peter Ölveczky\textsuperscript{1,2}

\textsuperscript{1}University of Oslo
\textsuperscript{2}University of Illinois at Urbana-Champaign

Thanks to Indranil Gupta (UIUC) and UIUC’s Center for Assured Cloud Computing
Goal

Cloud computing also when data consistency critical

- e-commerce, banking, medical systems
Part I: Formalizing Megastore
Data in the Cloud

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

- Availability and scalability: data must be replicated

(Figure from http://flux7.com/blogs/nosql/cap-theorem-why-does-it-matter/)
“Eventual consistency” OK for some applications
Eventual Consistency

“Eventual consistency” OK for some applications
Eventual Consistency

“Eventual consistency” OK for some applications
Databases traditionally provide **ACID transactions**

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

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

Little support for ACID transactions in cloud computing
Megastore:

- Google’s wide-area replicated data store
- Key part of Google’s cloud infrastructure
- 3 billion write and 20 billion read transactions daily (2011)
- Adds (limited) transactions to wide-area replicated data stores
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
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
Figure 1: Scalable Replication

(figures from J. Baker et al., “Megastore: Providing Scalable, Highly Available Storage for Interactive Services”)
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
Our Work

- Developed and formalized [our version of the] Megastore approach in Real-Time Maude
  - first (public) formalization/detailed description of Megastore
- 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
- 56 rewrite rules (37 for fault tolerance features)
- Real-Time Maude simulation and model checking used extensively throughout development of (our) Megastore
Real-Time Maude: rewriting-logic-based language and analysis tool

- **Expressive** formalism
  - algebraic specification specify data types
  - rewrite rules specify local transitions
- **Object-oriented** specification of distributed real-time systems
Real-Time Maude: rewriting-logic-based language and analysis tool

- **Expressive** formalism
  - algebraic specification specify data types
  - rewrite rules specify local transitions

- **Object-oriented** specification of distributed real-time systems

- **Formal analysis:**
  - simulation
  - reachability analysis
  - LTL model checking
  - Timed CTL model checking
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:

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

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

\[
\text{ACC-LEADER-REQ-MGS} \\
\text{if EIDSET := getEntityGroupIds(WRITEOPS) /\} \\
\quad \text{NEW-PAXOS-STATE := initiatePaxosState(EIDSET, TID, WRITEOPS,} \\
\quad \quad \quad \text{SID, RSTATE, EGROUPS) /\} \\
\quad \text{(createAcceptLeaderMessages(SID, NEW-PAXOS-STATE))} => \text{ACC-LEADER-REQ-MGS} \\
\]
A Rewrite Rule (cont.)

getchEntityGroupIds(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 (TID’ LP SID OL) represents the candidate log entry, containing the transaction identifier TID’, the log position LP, the proposed leader site SID, and the list of update operations OL.

crl [rcvAcceptAllReq] :
  (msg acceptAllReq(TID, EG, (TID’ LP SID OL), PROPNUM) from SENDER to 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)

<table>
<thead>
<tr>
<th>Network Delays</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></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:

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

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

  \[
  \text{eq} \ {\{\text{SYSTEM}\} \models \text{entityGroupsEqualOrInvalid} = \text{true} \ [\text{owise}] .}
  \]
**Model Checking (II)**

- **All replicas same unless coordinator invalidated:**
  
  \[
  \text{op entityGroupsEqualOrInvalid : } \rightarrow \text{Prop [ctor] .}
  \]

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

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

- **Execution serializable:**

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

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

- **All replicas same unless coordinator invalidated:**
  
  \[
  \text{op entityGroupsEqualOrInvalid} : \rightarrow \text{Prop} [\text{ctor}].
  \]

  \[
  \text{ceq } \{ < S_1 : \text{Site} | \text{coordinator} : \text{egl}p(\text{EG}_1, \text{LP}) ; \text{EGLP}, \text{entityGroups} : \\
  < \text{EG}_1 : \text{EntityGroup} | \text{entitiesState} : \text{ES}_1 > \text{EGS}_1 > \\
  < S_2 : \text{Site} | \text{coordinator} : \text{egl}p(\text{EG}_1, \text{LP}) ; \text{EGLP}, \text{entityGroups} : \\
  < \text{EG}_1 : \text{EntityGroup} | \text{entitiesState} : \text{ES}_2 > \text{EGS}_2 > \\
  \text{REST} \} \models \text{entityGroupsEqual} = \text{false if ES}_1 = \neq \text{ES}_2.
  \]

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

- **Execution serializable:**

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

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

- **Desired property:**

  \[
  <> [] (\text{allTransFinished} \land \text{entityGroupsEqualOrInvalid} \land \text{transLogsEqualOrInvalid} \land \text{isSerializable})
  \]
**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

<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></td>
<td></td>
<td></td>
<td>1367</td>
</tr>
<tr>
<td>19, 80</td>
<td></td>
<td></td>
<td>50, 200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20, 100</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>10, 50, 200</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1164</td>
</tr>
<tr>
<td>40, 80</td>
<td>872</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20, 20, 110</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>10, 30, 80</td>
<td>241</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30, 60, 120</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>3</td>
<td>20, 30, 40</td>
<td>1</td>
<td>{30, 80}</td>
<td>DNF</td>
</tr>
<tr>
<td>{30, 60, 120}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{10, 30, 80},and</td>
<td>3</td>
<td>20, 30, 40</td>
<td>1</td>
<td>60</td>
<td>DNF</td>
</tr>
<tr>
<td>{30, 60, 120}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part II: Megastore-CGC: extending Megastore
Our Work: Motivation

Goal

*Cloud computing* also when *data consistency* critical

- Sometimes transactions *must* access *multiple* entity groups
Our Work: Motivation

Goal

Cloud computing also when data consistency critical

- Sometimes transactions must access multiple entity groups
- Megastore: consistency only if transactions access single entity group
Our Work: Motivation

Goal

Cloud computing also when data consistency critical

- Sometimes transactions must access multiple entity groups
- Megastore: consistency only if transactions access single entity group
- Our approach: extend Megastore with consistency for transactions accessing multiple entity groups
Our Work: Motivation

Goal

*Cloud computing also when data consistency critical*

- Sometimes transactions **must** access **multiple** entity groups
- **Megastore**: consistency only if transactions access **single** entity group
- Our approach: extend **Megastore** with consistency for transactions accessing **multiple** entity groups
  - must maintain **Megastore**’s
    - performance
    - strong **fault tolerance**
Our Idea

- Key observations:
  1. A Megastore site replicating different entity groups participates in all updates on those entity groups
  2. → site has implicit local ordering on those updates
**Our Idea**

- **Key observations:**
  1. A Megastore site replicating different entity groups participates in all updates on those entity groups.
  2. The site has implicit local ordering on those updates.

- **Our idea:**
  1. Select one such site and make its order explicit.
  2. This site validates transactions on multiple entity groups before commit.

-  
  - ordering class = set of entity groups
  - ordering site for each ordering class
Megastore-CGC piggybacks ordering and validation onto Megastore’s coordination protocol

- no additional messages for validation/commit!
- maintains Megastore’s performance and fault tolerance
Megastore-CGC piggybacks ordering and validation onto Megastore’s coordination protocol

- no additional messages for validation/commit!
- maintains Megastore’s performance and fault tolerance
- failover protocol when ordering site fails
Developing Megastore-CGC

- Developed Megastore-CGC using Real-Time Maude
- 72 (fairly complex) rewrite rules
- Real-Time Maude simulation and model checking used extensively throughout development of Megastore-CGC
Software Engineering Perspective (I)

- Developing a fault-tolerant protocol is difficult
- Typically: pseudo-code description $\rightarrow$ Java prototype
Software Engineering Perspective (I)

- Developing a fault-tolerant protocol is difficult
- Typically: pseudo-code description $\rightarrow$ Java prototype
  - many hours of "whiteboard analysis"
  - still incorrect
Software Engineering Perspective (II)

- Our approach: Formal test-driven development:
  1. Express requirements as LTL formulas
  2. Develop Real-Time Maude model
  3. Test model using Real-Time Maude simulation and model checking
  4. For failing tests, analyse the problem and go to step (2)
Software Engineering Perspective (II)

• Our approach: **Formal test-driven development:**
  1. Express requirements as LTL formulas
  2. Develop Real-Time Maude model
  3. Test model using Real-Time Maude simulation and model checking
  4. For failing tests, analyse the problem and go to step (2)

• Allowed **one person** to develop and validate complex protocol in short time
Software Engineering Perspective (III)

- With much less effort, the fault-tolerance features of Megastore-CGC are significantly more mature than similar protocols the first author has been working on.
- Model checking replaces most of the whiteboard analysis.
Performance Comparison using Real-Time Maude

- Simulating for 1000 seconds (no failures)
- Megastore:

<table>
<thead>
<tr>
<th></th>
<th>Commits</th>
<th>Aborts</th>
<th>Avg. latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>652</td>
<td>152</td>
<td>126</td>
</tr>
<tr>
<td>Site 2</td>
<td>704</td>
<td>100</td>
<td>118</td>
</tr>
<tr>
<td>RSite</td>
<td>640</td>
<td>172</td>
<td>151</td>
</tr>
</tbody>
</table>

- Megastore-CGC:

<table>
<thead>
<tr>
<th></th>
<th>Commits</th>
<th>Aborts</th>
<th>Val. aborts</th>
<th>Avg. latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>660</td>
<td>144</td>
<td>0</td>
<td>123</td>
</tr>
<tr>
<td>Site 2</td>
<td>674</td>
<td>115</td>
<td>15</td>
<td>118</td>
</tr>
<tr>
<td>RSite</td>
<td>631</td>
<td>171</td>
<td>10</td>
<td>150</td>
</tr>
</tbody>
</table>
Model checking scenarios

• 5 transactions (3 fixed, 2 with 2 start times), no failures, message delay 30 ms or 80 ms → 108,279 reachable states, 124 seconds

• 3 transactions (all with 2 start times), one site failure and fixed message delay → 1,874,946 reachable states, 6,311 seconds

• 3 transactions (all with 2 start times), fixed message delay and one message failure → 265,410 reachable states, 858 seconds
**Conclusion**

- Developed model of Megastore from limited information
  - *first* formal analysis of transactional data stores
CONCLUSION

• Developed model of Megastore from limited information
  • first formal analysis of transactional data stores
• Extended the Megastore approach to Megastore-CGC
  • consistency for transactions accessing multiple entity groups
  • performance and fault tolerance on par with Megastore
CONCLUSION

- Developed model of Megastore from limited information
  - first formal analysis of transactional data stores
- Extended the Megastore approach to Megastore-CGC
  - consistency for transactions accessing multiple entity groups
  - performance and fault tolerance on par with Megastore
- Detailed formal specification in Real-Time Maude
  - shorter development time
  - increased quality of specification
  - analyze performance and correctness

- First generic (?) cloud data management protocol with consistency for transactions across partitions
  - Google’s Spanner: similar features with advanced infrastructure
  - Real-Time Maude expressiveness key
  - model complex system
  - express complex requirements
CONCLUSION

- Developed model of Megastore from limited information
  - first formal analysis of transactional data stores
- Extended the Megastore approach to **Megastore-CGC**
  - consistency for transactions accessing multiple entity groups
  - performance and fault tolerance on par with Megastore
- Detailed formal specification in **Real-Time Maude**
  - shorter development time
  - increased quality of specification
  - analyze performance and correctness
- First generic (?) cloud data management protocol with consistency for transactions across partitions
  - Google’s **Spanner**: similar features with advanced infrastructure
Conclusion

- Developed model of Megastore from limited information
  - first formal analysis of transactional data stores
- Extended the Megastore approach to Megastore-CGC
  - consistency for transactions accessing multiple entity groups
  - performance and fault tolerance on par with Megastore
- Detailed formal specification in Real-Time Maude
  - shorter development time
  - increased quality of specification
  - analyze performance and correctness
- First generic (?) cloud data management protocol with consistency for transactions across partitions
  - Google’s Spanner: similar features with advanced infrastructure
- Real-Time Maude expressiveness key
  - model complex system
  - express complex requirements