Design and Validation of Cloud Storage Systems using Maude

Peter Csaba Ölveczky

University of Oslo
University of Illinois at Urbana-Champaign

Based on joint work with Jon Grov and members of UIUC’s Center for Assured Cloud Computing
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Google</td>
<td>$109,470m</td>
<td>$88,173m</td>
<td>+24%</td>
<td>AAA+</td>
</tr>
<tr>
<td>2</td>
<td>Apple</td>
<td>$107,141m</td>
<td>$145,918m</td>
<td>-27%</td>
<td>AAA</td>
</tr>
<tr>
<td>3</td>
<td>Amazon</td>
<td>$106,369m</td>
<td>$69,642m</td>
<td>+53%</td>
<td>AAA-</td>
</tr>
<tr>
<td>4</td>
<td>AT&amp;T</td>
<td>$87,016m</td>
<td>$59,904m</td>
<td>+45%</td>
<td>AAA-</td>
</tr>
<tr>
<td>5</td>
<td>Microsoft</td>
<td>$76,265m</td>
<td>$67,258m</td>
<td>+13%</td>
<td>AAA</td>
</tr>
<tr>
<td>6</td>
<td>Samsung</td>
<td>$66,219m</td>
<td>$58,619m</td>
<td>+13%</td>
<td>AAA-</td>
</tr>
<tr>
<td>7</td>
<td>Verizon</td>
<td>$65,875m</td>
<td>$63,116m</td>
<td>+4%</td>
<td>AAA-</td>
</tr>
<tr>
<td>8</td>
<td>Walmart</td>
<td>$62,496m</td>
<td>$53,657m</td>
<td>+16%</td>
<td>AA+</td>
</tr>
<tr>
<td>9</td>
<td>Facebook</td>
<td>$61,998m</td>
<td>$34,002m</td>
<td>+82%</td>
<td>AAA</td>
</tr>
<tr>
<td>10</td>
<td>ICBC</td>
<td>$47,832m</td>
<td>$36,334m</td>
<td>+32%</td>
<td>AAA</td>
</tr>
</tbody>
</table>
Cloud computing systems store/retrieve large amounts of data.

Some Banks Are Heading To The Cloud -- More Are Planning To
Availability

Data should always be available
  - network/site failures, network congestion, scheduled upgrades
  -> data must be replicated
Availability

- Data should always be available
  - network/site failures, network congestion, scheduled upgrades
  - data must be replicated

- Large and growing data
  - Facebook (2014): 300 petabytes data; 350M photos uploaded every day
  - data must be partitioned
Consistency in Replicated Systems

Consistency: All replicas of a data item should have same value

Figure by Jiaqing Du
“CAP Theorem”

Data consistency + partition tolerance + availability impossible

(Figure from http://flux7.com/blogs/nosql/cap-theorem-why-does-it-matter/)
Trade-off

consistency level \leftrightarrow \text{ latency}
Eventual Consistency

- Weak consistency OK for some applications
Eventual Consistency

- Weak consistency OK for some applications
- ... but not others:
Designing Data Stores

- Complex systems
  - size
  - replication
  - concurrency
  - fault tolerance
Designing Data Stores

- Complex systems
  - size
  - replication
  - concurrence
  - fault tolerance
- Many hours of “whiteboard analysis”
Correctness: “hand proofs”
- error prone
- informal
- key assumptions implicit
- does not scale to nontrivial systems
Validating Data Store Designs

- **Correctness:** “hand proofs”
  - error prone
  - informal
  - key assumptions implicit
  - does not scale to nontrivial systems

- **Performance:** simulation tools, real implementations
  - additional artifact
  - cannot be used to reason about correctness
Our Approach: Formal Methods

Use formal methods to develop and validate designs
- define mathematical model of system
- use mathematical rules to analyze system
Our Approach: Formal Methods

Use formal methods to develop and validate designs
- define mathematical model of system
- use mathematical rules to analyze system

Find errors early!
**Formal system model $S$**
- precise mathematical model
- makes assumptions precise and explicit
- amenable to mathematical analysis
• **Formal system model** $S$
  - precise mathematical model
  - makes assumptions precise and explicit
  - amenable to mathematical analysis

• **Formal property specification** $P$
  - precise description of consistency model
  - can check whether $S \models P$
• **Formal system model** $S$
  ▶ precise mathematical model
  ▶ makes assumptions precise and explicit
  ▶ amenable to mathematical analysis

• **Formal property specification** $P$
  ▶ precise description of consistency model
  ▶ can check whether $S \models P$

• What about **performance analysis**?
Need:

- expressive and intuitive modeling language
Using Formal Methods (II): Software Engineering Perspective

Need:

- expressive and intuitive modeling language
- expressive and intuitive property specification language
Need:

- expressive and intuitive modeling language
- expressive and intuitive property specification language
- automatically check whether design satisfies property
  - quick and extensive feedback
  - saves days of whiteboard analysis
  - “extensive and automatic test suite”
Need:

- **expressive and intuitive modeling** language
- **expressive and intuitive property specification** language
- automatically check whether design satisfies property
  - quick and extensive feedback
  - saves days of whiteboard analysis
  - “extensive and automatic test suite”
- design model also for **performance analysis**!
  - no new artifact for performance analysis
Difficult challenges:

- intuitive
- expressive
- useful automatic analyses
- both correctness and performance analysis
- complex properties to check
- mature tool support
- real-time and probabilistic features
Our Framework: Rewriting Logic

- Rewriting logic: equations and rewrite rules
  - expressive
  - simple/intuitive
  - object-oriented

Maude tool:
- simulation
- temporal logic model checking
  ⋆ expressive property specification language

Extensions:
- real-time systems
- probabilistic systems
Our Framework: Rewriting Logic

- Rewriting logic: equations and rewrite rules
  - expressive
  - simple/intuitive
  - object-oriented

- Maude tool:
  - simulation
  - temporal logic model checking
    - expressive property specification language

Extensions:
- real-time systems
- probabilistic systems
Our Framework: Rewriting Logic

- **Rewriting logic:** equations and rewrite rules
  - expressive
  - simple/intuitive
  - object-oriented

- **Maude tool:**
  - simulation
  - temporal logic model checking
    - expressive property specification language

- **Extensions:**
  - real-time systems
  - probabilistic systems
Models can be developed quickly
• Models can be developed quickly
• **Simulation** gives quick feedback (**rapid prototyping**)
Models can be developed quickly

Simulation gives quick feedback (rapid prototyping)

Model checking: analyze all behaviors from one initial state

formal test-driven development: “test-driven development approach where many complex scenarios can be quickly tested by model checking”
What about performance analysis?
What about performance analysis?

1. (Randomized) simulations
What about performance analysis?

1. (Randomized) simulations
2. Probabilistic analysis (using PVeStA)
   - statistical model checking
Same artifact for:

- precise system description
- rapid prototyping
- extensive testing
- correctness analysis
- performance estimation
Case Study I

Modeling, Analyzing, and Extending Megastore

Joint work with Jon Grov (U. Oslo)
Megastore:

- Google’s wide-area replicated data store
- 3 billion write and 20 billion read transactions daily (2011)
Megastore: Key Ideas (I)

Data divided into entity groups

- Peter’s email
- Books on rewriting logic
- Narciso’s documents

(Figure from http://cse708.blogspot.jp/2011/03/megastore-providing-scalable-highly.html)
**Consistency** for transactions accessing a single entity group
- no guarantee if transaction reads *multiple* entity groups
[Developed and] formalized [our version of the] Megastore [approach] in Maude
Our Work

- Developed and formalized [our version of the] Megastore [approach] in Maude
  - first (public) formalization/detailed description of Megastore
Our Work

- Developed and formalized [our version of the] Megastore [approach] in Maude
  - first (public) formalization/detailed description of Megastore
- 56 rewrite rules (37 for fault tolerance features)
Performance Estimation

- Key performance measures:
  - average transaction latency
  - number of committed/aborted transactions
Performance Estimation

- Key performance measures:
  - average transaction latency
  - number of committed/aborted transactions

- Randomly generated transactions (rate 2.5 TPS)

- Network delays:
Performance Estimation

- Key performance measures:
  - average transaction latency
  - number of committed/aborted transactions

- 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>Madrid ↔ Paris</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Madrid ↔ 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:

<table>
<thead>
<tr>
<th></th>
<th>Avg. latency (ms)</th>
<th>Commits</th>
<th>Aborts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madrid</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>
Megastore-CGC: extending Megastore
Motivation

- Some transactions **must** access **multiple** entity groups
Motivation

- Some transactions **must** access **multiple** entity groups
- Our work: extend **Megastore** with consistency for transactions accessing **multiple** entity groups
Motivation

- Some transactions must access multiple entity groups
- Our work: extend Megastore with consistency for transactions accessing multiple entity groups
- Megastore-CGC piggybacks ordering and validation onto Megastore’s coordination protocol
  - no additional messages for validation/commit!
  - maintains Megastore’s performance and fault tolerance
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>Madrid</td>
<td>652</td>
<td>152</td>
<td>126</td>
</tr>
<tr>
<td>Paris</td>
<td>704</td>
<td>100</td>
<td>118</td>
</tr>
<tr>
<td>New York</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>Madrid</td>
<td>660</td>
<td>144</td>
<td>0</td>
<td>123</td>
</tr>
<tr>
<td>Paris</td>
<td>674</td>
<td>115</td>
<td>15</td>
<td>118</td>
</tr>
<tr>
<td>New York</td>
<td>631</td>
<td>171</td>
<td>10</td>
<td>150</td>
</tr>
</tbody>
</table>
Model checking scenarios

- 5 transactions, no failures, message delay 30 ms or 80 ms
  → 108,279 reachable states, 124 seconds

- 3 transactions, one site failure and fixed message delay
  → 1,874,946 reachable states, 6,311 seconds

- 3 transactions, fixed message delay and one message failure
  → 265,410 reachable states, 858 seconds
Case Study II

Work by Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, José Meseguer, Son Nguyen, Jatin Ganhotra (ICFEM’14, QEST’15)
Apache Cassandra

- Key-value data store originally developed at Facebook
- Used by Amadeus, Apple, CERN, IBM, Netflix, Facebook/Instagram, Twitter, ...
- Open source
Cassandra Overview

Read consistency either one, quorum, or all

![Diagram showing read consistency]

[Figures from http://www.slideshare.net/nuboat/cassandra-distributed-data-store]
Cassandra Overview

Read consistency either **one, quorum, or all**

Write consistency either **zero, one, quorum, or all**
Motivation

1. Formal model from 345K LOC
   ▶ allows experimenting with different optimizations/variations
2. Analyze basic property: eventual consistency
3. When/how often does Cassandra give stronger guarantees?
   ▶ strong consistency
   ▶ read-your-writes
4. Performance evaluation:
   ▶ compare PVeStA analyses with real implementations
### Formal Analysis with Multiple Clients

<table>
<thead>
<tr>
<th>Consistency Lvl.</th>
<th>Latency</th>
<th>ONE</th>
<th>QUORUM</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong L1</td>
<td>L1 (&lt;D1)</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>L2 (&lt;D1 &lt; L2 &lt; D2)</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>L3 (&lt;D2 &lt; L3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Consistency Lvl.</th>
<th>Latency</th>
<th>ONE</th>
<th>QUORUM</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual L1</td>
<td>L1 (&lt;D1)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>L2 (&lt;D1 &lt; L2 &lt; D2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>L3 (&lt;D2 &lt; L3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

**Conclusion**
- strong consistency depends on the latency between requests
- eventual consistency is guaranteed
Performance Estimation

Formal model + PVeStA vs. actual implementation

**Performance: Strong Consistency**

- **Statistical Model Checker**
  - (X axis =) Issuing Latency = time difference between the given read request and the latest write request
  - (Y axis =) Probability of a request satisfying that model

- **Real-deployed cluster**
  - QUORUM vs. ALL
P-Store

[N. Schiper, P. Sutra, and F. Pedone; IEEE SRDS’10]

- Replicated and partitioned data store
- Serializability
P-Store

[N. Schiper, P. Sutra, and F. Pedone; IEEE SRDS’10]

- Replicated and partitioned data store
- Serializability
- Atomic multicast orders concurrent transactions
P-Store

- Replicated and partitioned data store
- Serializability
- Atomic multicast orders concurrent transactions
- Group commitment for atomic commit
Atomic Multicast

Definition

Atomic Multicast: Consistent reception order of messages

- (a): any pair of nodes receive the same atomic-multicast messages in the same order

- (b): induced “global read order” must be acyclic

Example

A reads $m_1 < m_2$

B reads $m_2 < m_3$

C reads $m_3 < m_1$

satisfies (a) but not (b)
Atomic Multicast

**Definition**

Atomic Multicast: Consistent reception order of messages

- (a): any pair of nodes receive the same atomic-multicast messages in the same order
- (b): induced “global read order” must be acyclic

**Example**

- A reads $m_1 < m_2$
- B reads $m_2 < m_3$
- C reads $m_3 < m_1$

satisfies (a) but not (b)
Fundamental problem in distributed systems

Impose order on conflicting concurrent transactions
Fundamental problem in distributed systems

Impose order on conflicting concurrent transactions

Many algorithms for atomic multicast
Atomic Multicast in Maude (I)

- Fundamental problem in distributed systems
- **Impose order** on conflicting concurrent transactions
- Many algorithms for atomic multicast
- Define **generic atomic multicast** primitive in Maude
  - abstract
  - covers all possible receiving orders
Fundamental problem in distributed systems

**Impose order** on conflicting concurrent transactions

Many algorithms for atomic multicast

Define **generic atomic multicast** primitive in Maude
  - abstract
  - covers all possible receiving orders

Infrastructure stores (un)read AM messages
Atomic multicast message $M$:

```
rl [atomic-multicast] :
    < 0 : Node | msgToSend : M, receivers : OS >
=>
    < 0 : Node | ... >
(atomic-multicast $M$ from 0 to OS).
```
Atomic-multicast message \( M \):

\[
\begin{align*}
\text{rl} & \ [\text{atomic-multicast}] : \\
& \quad \langle 0 : \text{Node} \mid \text{msgToSend} : M, \text{receivers} : \text{OS} \rangle \\
& \quad \Rightarrow \\
& \quad \langle 0 : \text{Node} \mid \ldots \rangle \\
& \quad (\text{atomic-multicast } M \text{ from } 0 \text{ to } \text{OS}).
\end{align*}
\]

Read:

\[
\begin{align*}
\text{crl} & \ [\text{receiveAtomicMulticast}] : \\
& \quad (\text{msg } M \text{ from } O_2 \text{ to } 0) \\
& \quad \langle 0 : \text{Node} \mid \ldots \rangle \\
& \quad \text{AM-TABLE} \\
& \quad \Rightarrow \\
& \quad \langle 0 : \text{Node} \mid \ldots \rangle \\
& \quad \text{updateAM}(\text{MC}, 0, \text{AM-TABLE}) \\
& \quad \text{if } \text{okToRead}(\text{MC}, 0, \text{AM-TABLE}).
\end{align*}
\]
Analyzing P-Store

Find all reachable final states from init3:

Maude> (search init3 =>! C:Configuration .)

Solution 1
C:Configuration --> ...
< c1 : Client | pendingTrans : t1, txns : emptyTransList >
< c2 : Client | pendingTrans : t2, txns : emptyTransList >
< r1 : PStoreReplica | aborted : none,
committed : < t1 : Transaction | ... >
< r2 : PStoreReplica | aborted : none,
committed : < t2 : Transaction | ... >
...

sites validate transactions but client never gets result
Analyzing P-Store

Find all reachable final states from init3:

Maude> (search init3 =>! C:Configuration .)

Solution 1
C:Configuration --> ...
< c1 : Client | pendingTrans : t1, txns : emptyTransList >
< c2 : Client | pendingTrans : t2, txns : emptyTransList >
< r1 : PStoreReplica | aborted : none,
  committed : < t1 : Transaction | ... >
< r2 : PStoreReplica | aborted : none,
  committed : < t2 : Transaction | ... >
...

- sites validate transactions
- but client never gets result
Solution 5

...<r1 : PStoreReplica | aborted : none, committed : none,
submitted : < t1 : Transaction | ... >, ... >
<r2 : PStoreReplica | aborted : none,
committed : < t2 : Transaction| ... > ... >
Analyzing P-Store (cont.)

Solution 5
...
< r1 : PStoreReplica | aborted : none, committed : none,
  submitted : < t1 : Transaction | ... >, ... >
< r2 : PStoreReplica | aborted : none,
  committed : < t2 : Transaction | ... > ...

- Host does not validate t1 even when needed info known
Fixing P-Store

- Found the source of the errors
  - all replicas must be involved in voting and notification
    - not just write replicas
- Modeled and analyzed proposed corrected version
“P-Store verified”

- 3 significant errors found
- one confusing definition
- key assumption missing
Our Conclusions
Our Conclusions I

- Developed formal models of large industrial data stores
  - Google’s Megastore (from brief description)
  - Apache Cassandra (from 345K LOC and description)
  - P-Store (academic)
Our Conclusions I

- Developed formal models of large industrial data stores
  - Google’s Megastore (from brief description)
  - Apache Cassandra (from 345K LOC and description)
  - P-Store (academic)

- Automatic model checking analysis of consistency properties
Our Conclusions I

- Developed **formal** models of **large industrial** data stores
  - Google’s **Megastore** (from brief description)
  - Apache **Cassandra** (from 345K LOC and description)
  - **P-Store** (academic)

- Automatic **model checking** analysis of consistency properties

- Designed **own** transactional data stores
  - **Megastore-CGC**
  - variation of Cassandra
Our Conclusions I

- Developed **formal** models of **large industrial** data stores
  - Google’s Megastore (from brief description)
  - Apache Cassandra (from 345K LOC and description)
  - P-Store (academic)
- **Automatic model checking** analysis of consistency properties
- Designed **own** transactional data stores
  - Megastore-CGC
  - variation of Cassandra
- **Errors, ambiguities, missing assumptions** found in “verified” P-Store
Our Conclusions I

- Developed **formal** models of **large industrial** data stores
  - Google's **Megastore** (from brief description)
  - Apache **Cassandra** (from 345K LOC and description)
  - **P-Store** (academic)

- **Automatic model checking** analysis of consistency properties

- Designed **own** transactional data stores
  - Megastore-CGC
  - variation of Cassandra

- **Errors, ambiguities, missing assumptions** found in “verified” P-Store

- Maude/PVeStA **performance estimation** close to real implementations
Our “Software Engineering” Conclusions

- Quickly develop formal models/prototypes of complex systems
  - experiment with different design choices
Our “Software Engineering” Conclusions

- **Quickly** develop **formal** models/prototypes of **complex** systems
  - experiment with different design choices
- **Simulation** and **model checking** throughout design phase
  - model-checking-based-testing for subtle “corner cases”
  - replaces days of **whiteboard analysis**
  - too many scenarios for standard test-based development
  - catch bugs **early**!
Our “Software Engineering” Conclusions

- **Quickly** develop **formal** models/prototypes of **complex** systems
  - experiment with different design choices
- **Simulation** and **model checking throughout** design phase
  - model-checking-based-testing for subtle “corner cases”
  - replaces days of **whiteboard analysis**
  - too many scenarios for standard test-based development
  - catch bugs **early!**
- **Single artifact** for
  - system description
  - rapid prototyping
  - model checking
  - performance estimation
Our “Software Engineering” Conclusions

- **Quickly** develop **formal** models/prototypes of **complex** systems
  - experiment with different design choices

- **Simulation and model checking** throughout design phase
  - model-checking-based-testing for subtle “corner cases”
  - replaces days of **whiteboard analysis**
  - **too many scenarios** for standard test-based development
  - catch bugs **early**!

- **Single artifact** for
  - system description
  - rapid prototyping
  - model checking
  - performance estimation

- Megastore and Megastore-CGC modeler had **no formal methods experience**
Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

By Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff

How Amazon Web Services Uses Formal Methods
Amazon Web Services

Amazon Web Services (AWS):

- world’s largest cloud computing service provider
- more profitable than Amazon’s retail business
Amazon Web Services

- Amazon Web Services (AWS):
  - world’s largest cloud computing service provider
  - more profitable than Amazon’s retail business

- Amazon Simple Storage Service (S3)
  - stores > 3 trillion objects
  - 99.99% availability of objects
  - > 1 million requests per second

- DynamoDB data store
Amazon Web Services and Formal Methods

- **Formal methods** used **extensively** at AWS during design of S3, DynamoDB, ...
- Used Lamport's TLA+
  - model checking
Model checking finds “corner case” bugs that would be hard to find with standard industrial methods:
Model checking finds “corner case” bugs that would be hard to find with standard industrial methods:

- “We have found that standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex fault-tolerant systems.”

Experiences at Amazon WS

Model checking finds “corner case” bugs that would be hard to find with standard industrial methods:

- “the model checker found a bug that could lead to losing data [...]This was a very subtle bug; the shortest error trace exhibiting the bug included 35 high-level steps. [...] The bug had passed unnoticed through extensive design reviews, code reviews, and testing.”
A formal specification is a valuable precise description of an algorithm:
A formal specification is a valuable precise description of an algorithm:

- “the author is forced to think more clearly, helping eliminating “hand waving,” and tools can be applied to check for errors in the design, even while it is being written. In contrast, conventional design documents consist of prose, static diagrams, and perhaps psuedo-code in an ad hoc untestable language.”
A formal specification is a valuable precise description of an algorithm:

- “Talk and design documents can be ambiguous or incomplete, and the executable code is much too large to absorb quickly and might not precisely reflect the intended design. In contrast, a formal specification is precise, short, and can be explored and experimented on with tools.”
Formal methods are surprisingly feasible for mainstream software development and give good return on investment:
Experiences at Amazon WS III

Formal methods are surprisingly feasible for mainstream software development and give good return on investment:

- “In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code. Our experience with TLA+ shows this perception to be wrong. [...] Amazon engineers have used TLA+ on 10 large complex real-world systems. In each, TLA+ has added significant value. [...] Engineers have been able to learn TLA+ from scratch and get useful results in two to three weeks.”
Experiences at Amazon WS III

Formal methods are surprisingly feasible for mainstream software
development and give good return on investment:

“Using TLA+ in place of traditional proof writing would thus likely
have improved time to market, in addition to achieving greater
confidence in the system’s correctness.”
Experiences at Amazon WS III

Quick and easy to experiment with different design choices:
Quick and easy to experiment with different design choices:

- “We have been able to make innovative performance optimizations [...] we would not have dared to do without having model-checked those changes. A precise, testable description of a system becomes a what-if tool for designs.”
Experiences at Amazon WS: Limitations

TLA+ did/could not analyze performance degradation
Maude vs TLA+

Maude should be better suited!

- more intuitive and expressive specification language
  - OO
  - hierarchical states
  - dynamic object/message creation/deletion
  - ...

- Support for **real-time** and **probabilistic** systems

- Also for **performance estimation**!
Conclusions at Amazon

key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.

- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.

- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.
Formal methods can be an efficient way to
  ▶ design
  ▶ test
  ▶ describe
  ▶ validate correctness and performance
  ▶ experiment with different design choices

industrial state-of-the-art fault-tolerant distributed systems also for
non-experts
Formal methods can be an efficient way to
- design
- test
- describe
- validate correctness and performance
- experiment with different design choices

industrial state-of-the-art fault-tolerant distributed systems also for non-experts

Maude suitable modeling language and analysis toolset