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


ALPHA
3 DEF
2 ABC
6 MNO
1 QZ.
5 JKL
9 WXY
4 GHI
8 TUV
#
7 PRS
0 .SP

Global 500 2017

The most valuable brands of 2017

1 Rank 2017: **1** 2016: **2** ↑
BV 2017: **$109,470m**
BV 2016: **$88,173m** +24%
Brand Rating: **AAA+**

2 Rank 2017: **2** 2016: **1** ↓
BV 2017: **$107,141m**
BV 2016: **$145,918m** -27%
Brand Rating: **AAA**

3 Rank 2017: **3** 2016: **3** →
BV 2017: **$106,369m**
BV 2016: **$69,642m** +53%
Brand Rating: **AAA-**

4 Rank 2017: **4** 2016: **6** ↑
BV 2017: **$87,016m**
BV 2016: **$59,904m** +45%
Brand Rating: **AAA**

5 Rank 2017: **5** 2016: **4** ↓
BV 2017: **$76,265m**
BV 2016: **$ 67,258m** +13%
Brand Rating: **AAA**

6 Rank 2017: **6** 2016: **7** ↑
BV 2017: **$ 66,219m**
BV 2016: **$ 58,619m** +13%
Brand Rating: **AAA-**

7 Rank 2017: **7** 2016: **5** ↓
BV 2017: **$ 65,875m**
BV 2016: **$ 63,116m** +4%
Brand Rating: **AAA-**

8 Rank 2017: **8** 2016: **8** →
BV 2017: **$ 62,496m**
BV 2016: **$ 53,657m** +16%
Brand Rating: **AA+**

9 Rank 2017: **9** 2016: **17** ↑
BV 2017: **$ 61,998m**
BV 2016: **$ 34,002m** +82%
Brand Rating: **AAA**

10 Rank 2017: **10** 2016: **13** ↑
BV 2017: **$ 47,832m**
BV 2016: **$ 36,334m** +32%
Brand Rating: **AAA**

Cloud Computing Data Stores

Cloud computing systems store/retrieve large amounts of data

Forbes / Tech

Top 20 St

JUN 26, 2014 @ 02:17 PM 26,382 VIEWS

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

Availability

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

$\longrightarrow$ data must be replicated

Availability

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

  $\longrightarrow$ data must be replicated
- Large and growing data
  - Facebook (2014): 300 petabytes data; 350M photos uploaded every day

  $\longrightarrow$ data must be partitioned

Consistency in Replicated Systems

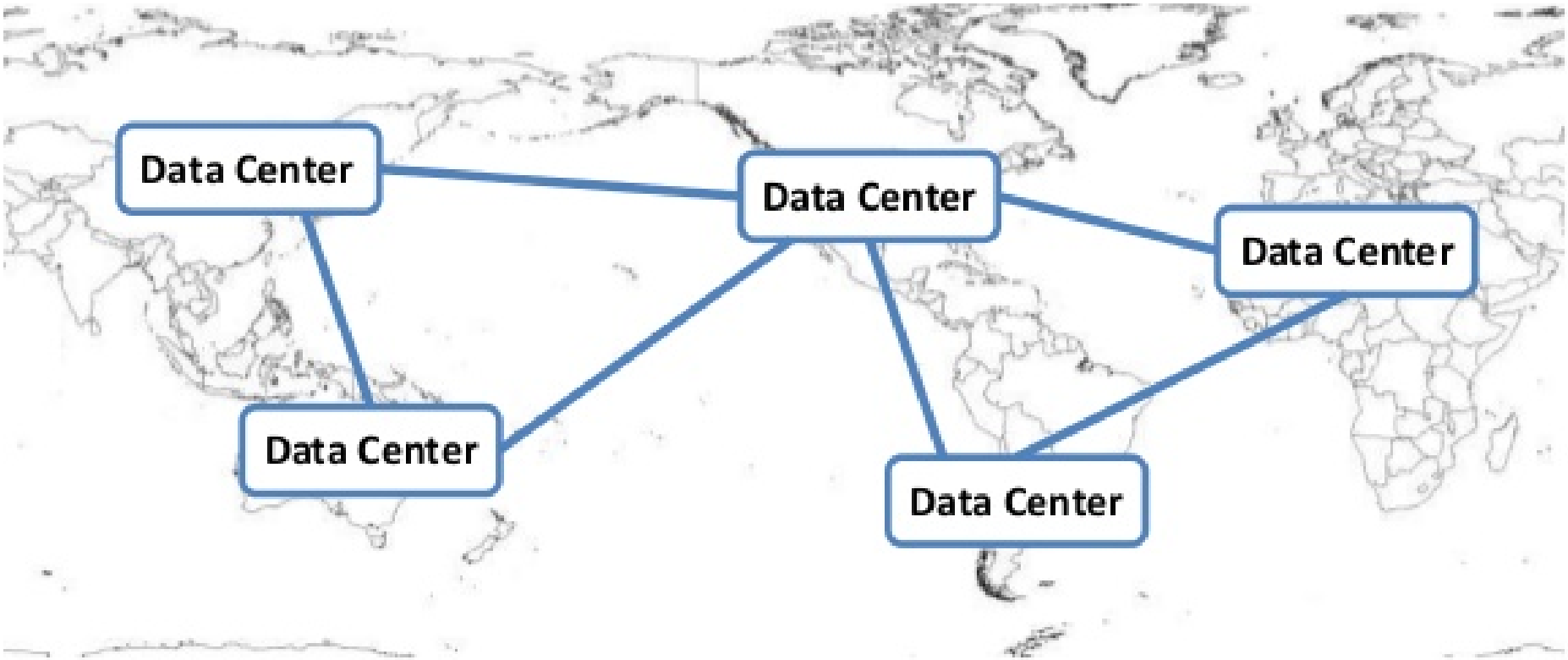
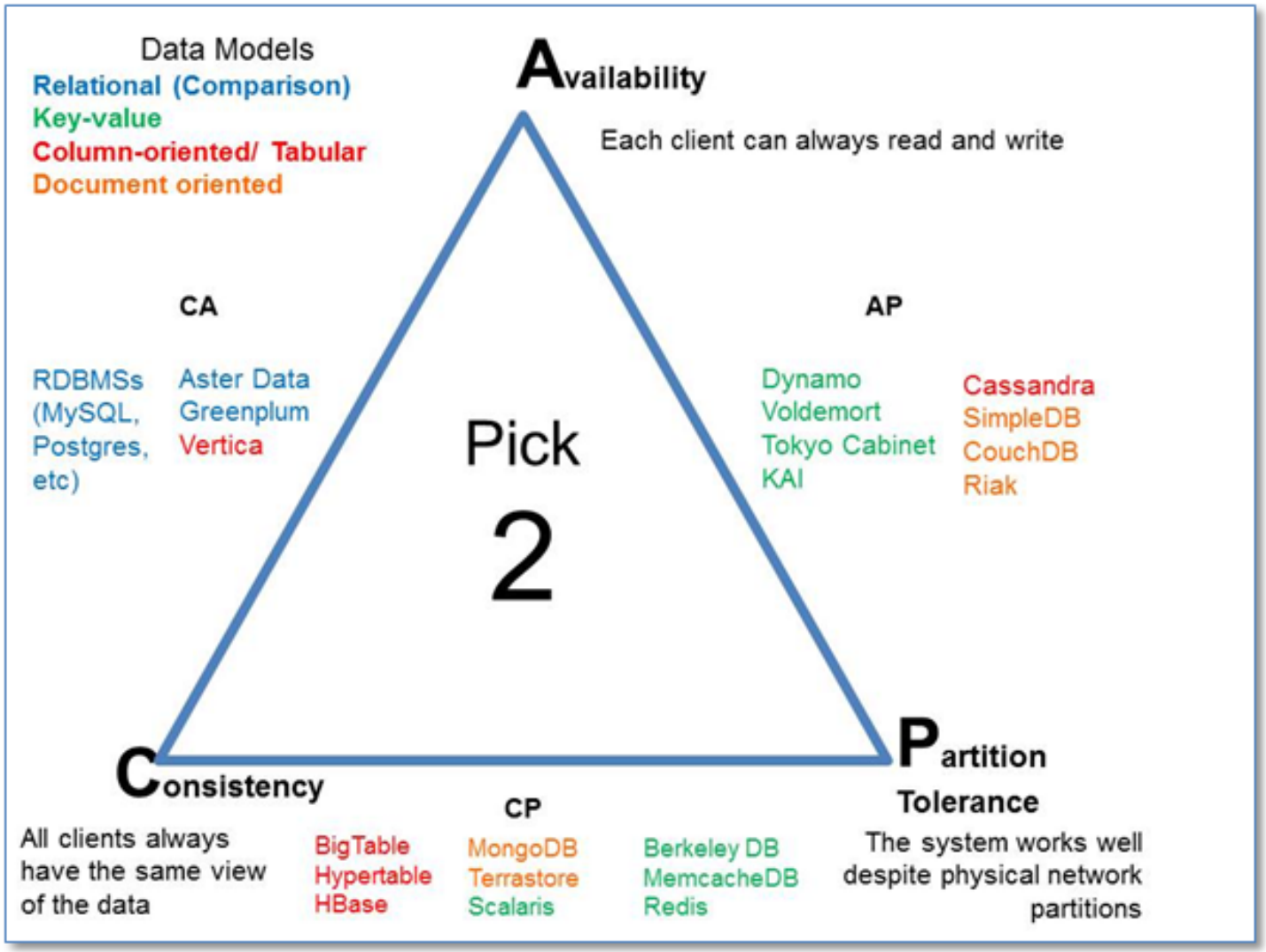


Figure by Jiaqing Du

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

“CAP Theorem”

Data consistency + partition tolerance + availability impossible



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

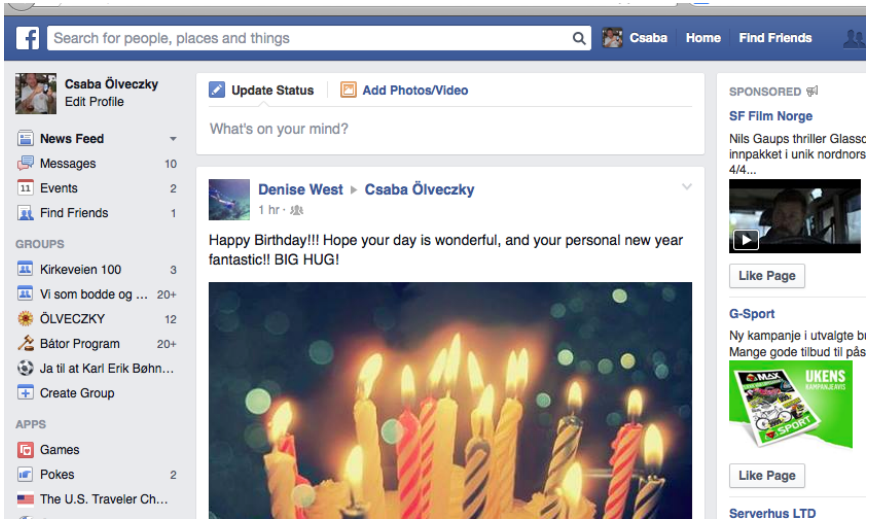
Slightly Different View

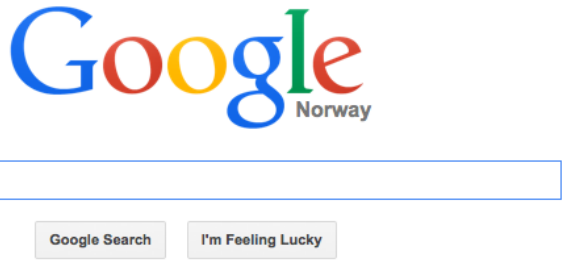
Trade-off

consistency level $\longleftrightarrow$ latency

Eventual Consistency

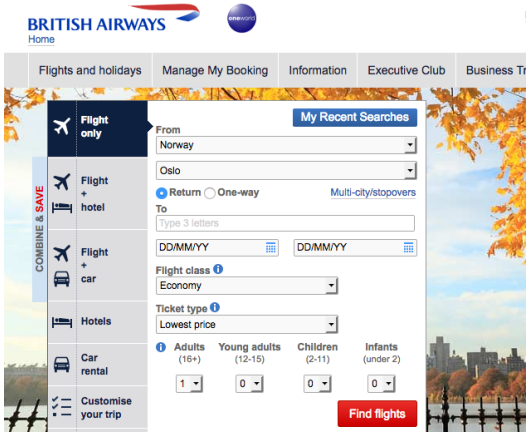
- Weak consistency OK for some applications

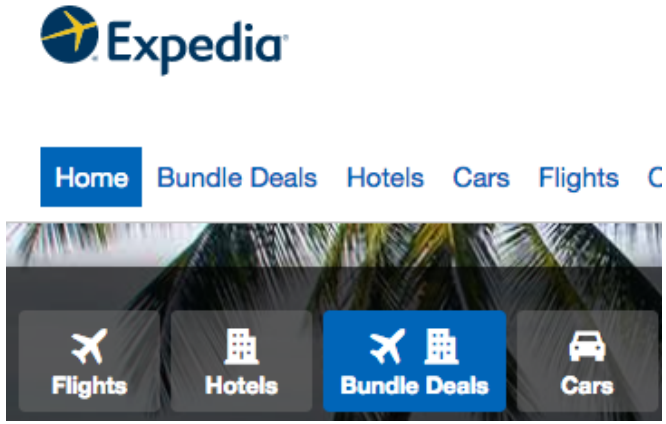




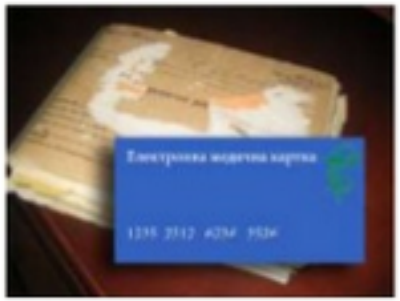
Eventual Consistency

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





Electronic patient registry



Electronic medical records

Designing Data Stores

- Complex systems
  - size
  - replication
  - concurrence
  - fault tolerance

Designing Data Stores

- Complex systems
  - size
  - replication
  - concurrence
  - fault tolerance
- Many hours of "whiteboard analysis"

Validating Data Store Designs

- 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!

Using Formal Methods (I): Validation Perspective

- Formal system model S
  - precise mathematical model
  - makes assumptions precise and explicit
  - amenable to mathematical analysis

Using Formal Methods (I): Validation Perspective

- 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 $S \models P$

Using Formal Methods (I): Validation Perspective

- 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?

Using Formal Methods (II): Software Engineering Perspective

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

Using Formal Methods (II): Software Engineering Perspective

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"

Using Formal Methods (II): Software Engineering Perspective

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

Which Formal Language/Tool?

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

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

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

Maude: Software Engineering Perspective I

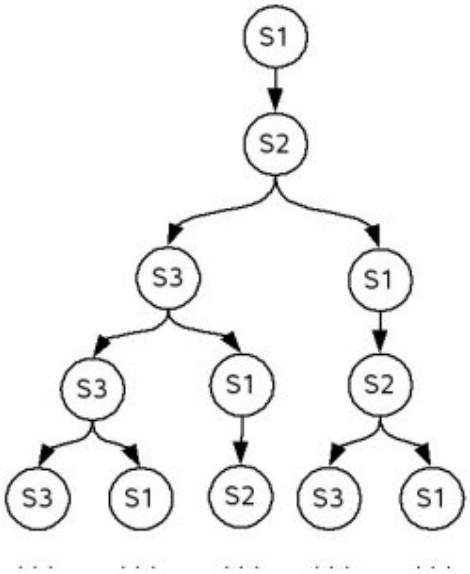
- Models can be developed quickly

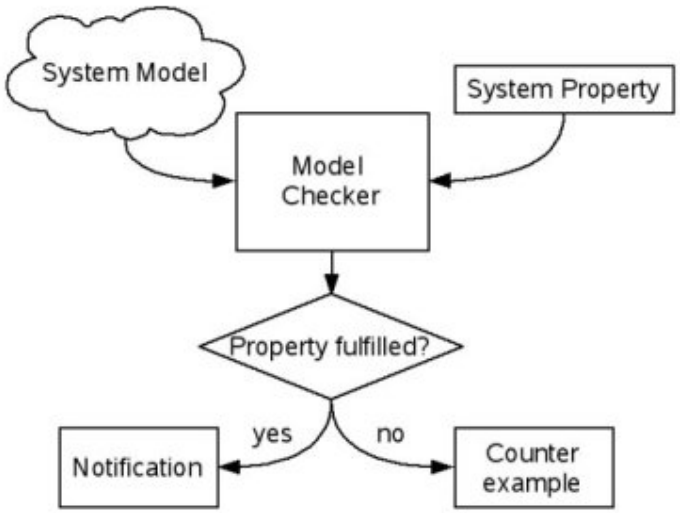
Maude: Software Engineering Perspective I

- Models can be developed quickly
- Simulation gives quick feedback (rapid prototyping)

Maude: Software Engineering Perspective I

- Models can be developed quickly
- Simulation gives quick feedback (rapid prototyping)
- Model checking: analyze all behaviors from one initial state





http://embsys.technikum-wien.at/projects/decs/verification/formalmethods.php

- formal test-driven development: "test-driven development approach where many complex scenarios can be quickly tested by model checking"

Maude: Software Engineering Perspective (cont.)

What about performance analysis?

Maude: Software Engineering Perspective (cont.)

What about performance analysis?

1. (Randomized) simulations

Maude: Software Engineering Perspective (cont.)

What about performance analysis?

1. (Randomized) simulations
2. Probabilistic analysis (using PVeStA)
   - statistical model checking

Maude: Software Engineering Perspective (cont.)

Same artifact for:

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

Case Study I

Modeling, Analyzing, and Extending Megastore

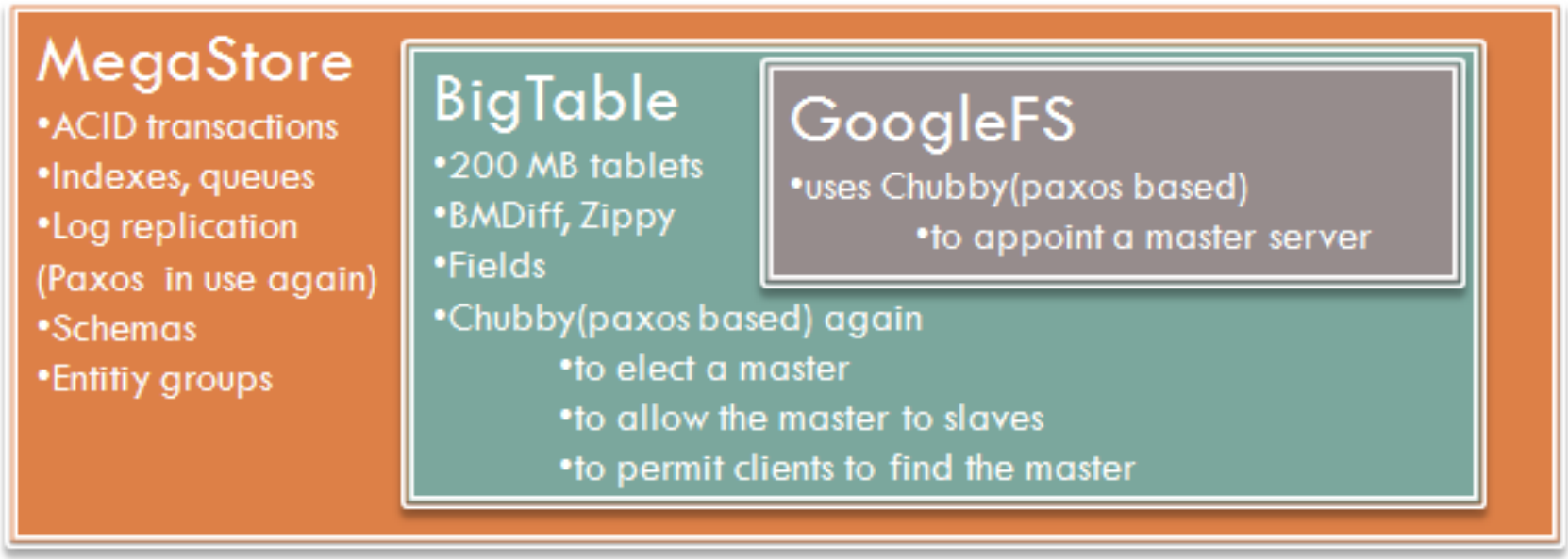
Joint work with Jon Grov (U. Oslo)

Megastore

Megastore:

- Google's wide-area replicated data store
- 3 billion write and 20 billion read transactions daily (2011)

Megastore: Key Ideas (I)



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

Data divided into entity groups

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

Megastore: Key Ideas (II)

- Consistency for transactions accessing a single entity group
  - no guarantee if transaction reads multiple entity groups

Our Work

- [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:

	30%	30%	30%	10%
Madrid $\leftrightarrow$ Paris	10	15	20	50
Madrid $\leftrightarrow$ New York	30	35	40	100
Paris $\leftrightarrow$ New York	30	35	40	100

Performance Estimation

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

	30%	30%	30%	10%
Madrid $\leftrightarrow$ Paris	10	15	20	50
Madrid $\leftrightarrow$ New York	30	35	40	100
Paris $\leftrightarrow$ New York	30	35	40	100

- Simulating for 200 seconds:

	Avg. latency (ms)	Commits	Aborts
Madrid	218	109	38
New York	336	129	16
Paris	331	116	21

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

Performance Comparison using Real-Time Maude

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

	Commits	Aborts	Avg. latency (ms)
Madrid	652	152	126
Paris	704	100	118
New York	640	172	151

- Megastore-CGC:

	Commits	Aborts	Val. aborts	Avg.latency (ms)
Madrid	660	144	0	123
Paris	674	115	15	118
New York	631	171	10	150

Model Checking Megastore-CGC

Model checking scenarios

- 5 transactions , no failures, message delay 30 ms or 80 ms
  $\longrightarrow$ 108,279 reachable states, 124 seconds
- 3 transactions, one site failure and fixed message delay
  $\longrightarrow$ 1,874,946 reachable states, 6,311 seconds
- 3 transactions, fixed message delay and one message failure
  $\longrightarrow$ 265,410 reachable states, 858 seconds

Case Study II

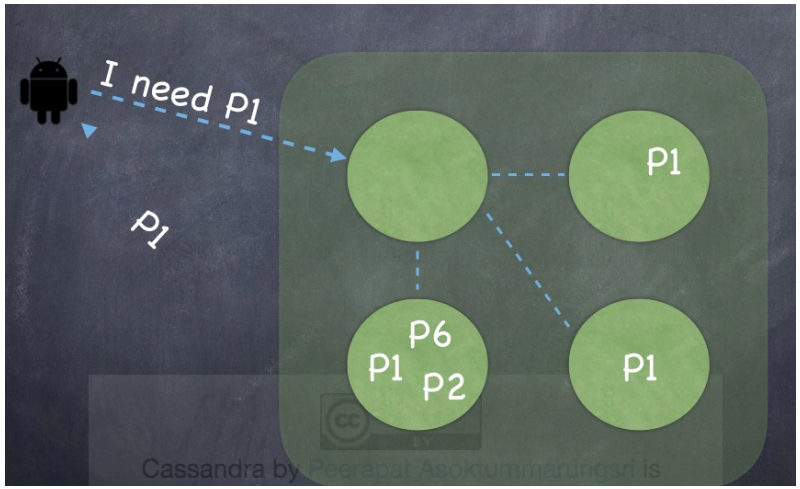
Work by Si Liu, Muntasir Raihan Rahman, Stephen Skeirik, Indranil Gupta, José Meseguer, Son Nguyen, Jatin Ganhotra (ICFEM'14, QEST'15)

Apache Cassandra

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

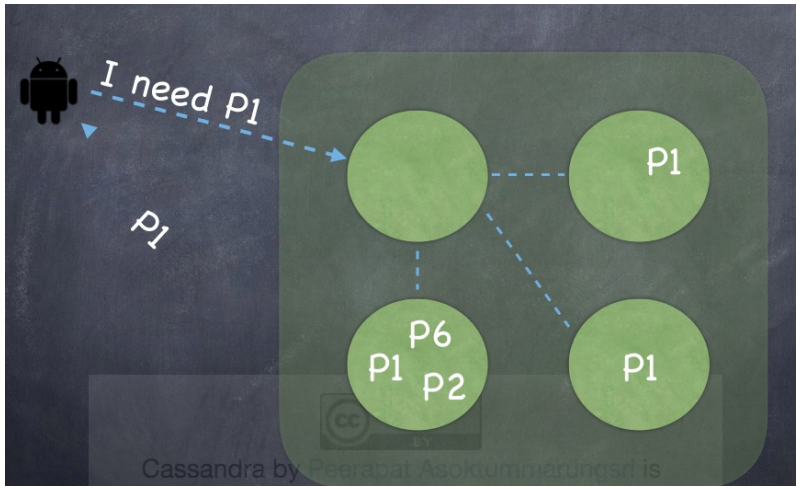
Cassandra Overview

Read consistency either one, quorum, or all

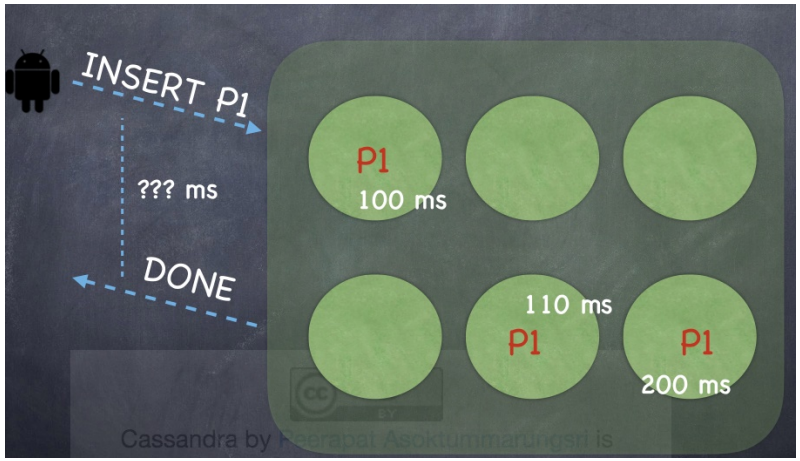


Cassandra Overview

Read consistency either one, quorum, or all



Write consistency either zero, one, quorum, or all



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

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

Strong

Latency \ Consistency Lv.	ONE	QUORUM	ALL
L1 (L1<D1)	×	×	×
L2 (D1<L2<D2)	×	×	×
L3 (D2<L3)	✓	✓	✓

Relation between issuing latency and message delays

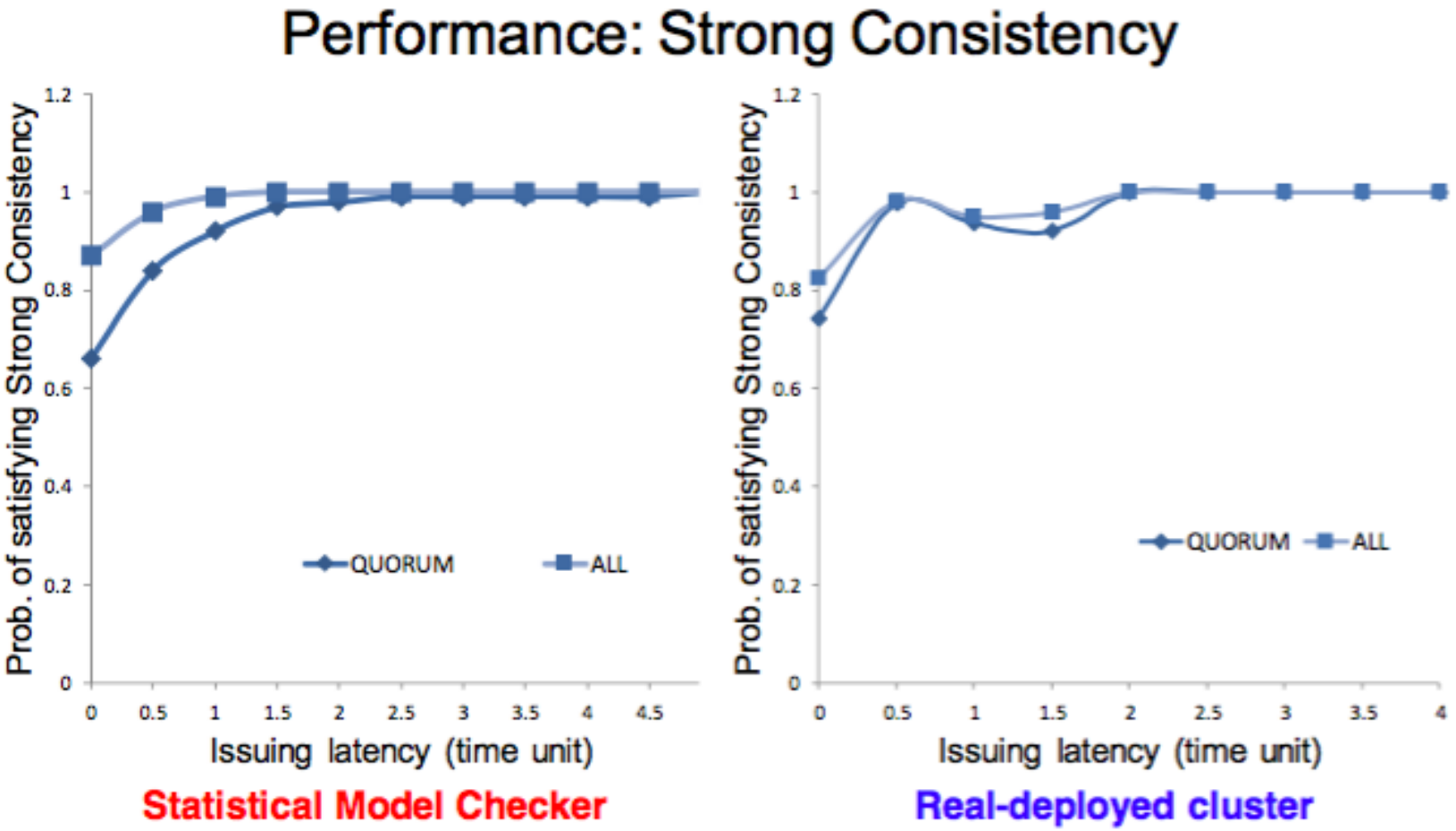
Eventual

Latency \ Consistency Lv.	ONE	QUORUM	ALL
L1 (L1<D1)	✓	✓	✓
L2 (D1<L2<D2)	✓	✓	✓
L3 (D2<L3)	✓	✓	✓

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

Performance Estimation

Formal model + PVeStA vs. actual implementation



- (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

P-Store

P-Store [N. Schiper, P. Sutra, and F. Pedone; IEEE SRDS'10]

- Replicated and partitioned data store
- Serializability

P-Store

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

P-Store [N. Schiper, P. Sutra, and F. Pedone; IEEE SRDS'10]

- 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

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 in Maude (I)

- Fundamental problem in distributed systems
- Impose order on conflicting concurrent transactions

Atomic Multicast in Maude (I)

- 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

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
- Infrastructure stores (un)read AM messages

My Work: Atomic Multicast in Maude (II)

- Atomic-multicast message M:

```
rl [atomic-multicast] :
   < O : Node | msgToSend : M, receivers : OS >
 =>
   < O : Node | ... >
   (atomic-multicast M from O to OS) .
```

My Work: Atomic Multicast in Maude (II)

- Atomic-multicast message M:

```
rl [atomic-multicast] :
   < O : Node | msgToSend : M, receivers : OS >
 =>
   < O : Node | ... >
   (atomic-multicast M from O to OS) .
```

- Read:

```
crl [receiveAtomicMulticast] :
    (msg M from O2 to O)
    < O : Node | ... >
    AM-TABLE
  =>
    < O : Node | ... >
    updateAM(MC, O, AM-TABLE)
  if okToRead(MC, O, AM-TABLE) .
```

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 | ... >
...
```

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 (cont.)

```
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 Summary

Algorithm $\mathcal{A}_{ge}$
A Genuine Certification Protocol - Code of site s

```
1: Initialization
2:    Votes ← ∅

3: function ApplyUpdates(T)
4:    foreach ∀(k, v) ∈ T.up : k ∈ Items(s) do
5:       let ts be Version(k, s)
6:       w_T[k, v, ts + 1]                          {write to the database}

7: function Certify(T)
8:    return ∀(k, ts) ∈ T.rs s.t. k ∈ Items(s) : ts = Version(k, s)

9: To submit transaction T                                        {Task 1}
10:   A-MCast(T) to Replicas(T)                  {Executing → Submitted}

11: When receive(VOTE, T.id, vote) from s'                        {Task 2}
12:   Votes ← Votes ∪ (T.id, s', vote)

13: When A-Deliver(T)                                             {Task 3}
14:   if T is local then
15:      if Certify(T) then
16:         ApplyUpdates(T)
17:         commit T                                {Submitted → Committed}
18:      else abort T                                 {Submitted → Aborted}
19:   else
20:      if ∃(k, -) ∈ T.rs : k ∈ Items(s) then
21:         Votes ← Votes ∪ (T.id, s, Certify(T))
22:         send(VOTE, T.id, Certify(T)) to all s' in WReplicas(T) s.t.
                                                  s' ∉ group(s)
23:      if s ∈ WReplicas(T) then
24:         wait until ∃VQ ∈ VQ(T) :
                        ∀s' ∈ VQ : (T.id, s', -) ∈ Votes
25:         if ∀s' ∈ VQ : (T.id, s', yes) ∈ Votes then
26:            ApplyUpdates(T)
27:            commit T                             {Submitted → Committed}
28:         else abort T                              {Submitted → Aborted}
29:   if s ∈ WReplicas(T) then send T's outcome to Proxy(T)
```

- "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

DOI:10.1145/2699417

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 Lamports TLA+
  - model checking

Experiences at Amazon WS

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

Experiences at Amazon WS

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."

Experiences at Amazon WS II

A formal specification is a valuable precise description of an algorithm:

Experiences at Amazon WS II

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."

Experiences at Amazon WS II

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."

Experiences at Amazon WS III

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:

Experiences at Amazon WS III

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.**

Take Away from Talk

- 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

Take Away from Talk

- 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