Formal Design of Cloud Computing Systems in Maude

José Meseguer

Department of Computer Science
University of Illinois at Urbana-Champaign, USA

Abstract. Cloud computing systems are complex distributed systems whose design is challenging for two main reasons: (1) since they are distributed systems, a correct design is very hard to achieve by testing alone; and (2) cloud computing applications have high availability and performance requirements; but these are hard to measure before implementation and hard to compare between different implementations. This paper summarizes our experience in using formal specification in Maude and model checking analysis to quickly explore the design space of a cloud computing system to achieve a high quality design that: (1) has verified correctness guarantees; (2) has better performance properties than other design alternatives so explored; (3) can be achieved before an actual implementation; and (4) can be used for both rapid prototyping and for automatic code generation.
Keywords: specification and verification of distributed systems, cloud computing, rewriting logic, Maude.

1 The Challenge of Cloud Computing

Cloud computing systems are used massively and need to meet high performance requirements such as as high availability and throughput, and low latency, even with network congestion and faults, and during software and hardware upgrades. Furthermore, for both high availability and fault tolerance, data has to be replicated. However, the *CAP theorem* [13] shows that it is impossible to simultaneously have high availability and strong consistency in replicated data stores. This means that, depending on the application, different tradeoffs need to be found in the design of a cloud computing system between consistency and performance. For example, for a social network a weak consistency notion such as "eventual consistency" may be acceptable in exchange for high performance, whereas a medical information system will clearly require stronger consistency notions, even at the cost of some losses in performance. Indeed, as explained in [14], there is a wide spectrum of consistency models to choose from. One of the most crucial tasks in the design of a cloud computing system is to achieve a good balance between good performance and consistency guarantees that are sufficient for the kinds of applications intended for the given system.

With some notable exceptions (see, e.g., [34]), in practice, cloud computing systems are often designed and built using only informal designs and only with

the aid of testing techniques. Also, only after a system has been for the most part built, do experimental evaluations become possible. Since furthermore, these distributed systems can be quite large (for example, Cassandra has about 345,000 lines of code) and fairly complex, all this means that: (i) subtle bugs can easily pass undetected; (ii) it may not be entirely clear what consistency and correctness guarantees can be given for the system; and (iii) it can be very costly to explore other design alternatives, since the cost of implementing them is too high.

All this also means that there is a good opportunity for formal methods to provide much needed analytic and predictive power for exploring cloud computing system designs *before they are built*. However, this is also a challenge since:

1. The formal methods employed must naturally support distributed system design and analysis.
2. The formal notations used should be easy to understand by system designers. Furthermore, they should be simple and concise enough to precisely capture design ideas at a high level in specifications orders of magnitude shorter than code. This then makes it easy to express alternative designs and to explore the practical impact of various design choices.
3. It should be possible to analyze correctness properties, if possible automatically, and to obtain counterexamples when such properties are violated.
4. Since for these systems high performance is as important as correctness, the formal analysis methods should also be able to provide not just "yes" or "no" answers to logical correctness questions, but also *quantitative* answers to performance questions.

It is worth stressing that point (2) is of great importance: only after having arrived at a good design is it meaningful to spend further efforts verifying in depth its properties. To put it perhaps more sharply, the exploration of a system's design and that of its logical and performance properties should happen *simultaneously* and, once a good design has thus been identified, its formal analysis should increase in depth. Also, all this should be done *before* actually building the systems, so that: (a) costly design errors are caught as early as possible; and (b) as much as possible is known about such a design, including its logical correctness properties and its estimated performance, before it is built.

2 Rewriting Logic and Maude

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ specifies a *concurrent system*, whose *states* are the elements of the algebraic data type $T_{\Sigma/E}$, and whose *concurrent transitions* are specified by the rewrite rules R. We have found rewriting logic particularly well suited for specifying cloud computing systems. Such systems can be naturally specified as configurations of distributed objects, often clients and servers, which communicate with each other through message passing. The sending and receiving of messages by such objects has a very natural formalization by means

of simple rewrite rules. In our experience, the rewrite rule formalism is easy to understand by network engineers and distributed system designers.

Maude [15] is a language implementing rewriting logic. Since a program in Maude is just a rewrite theory, Maude is a very simple language. However, Maude is both highly expressive and versatile and very high level, affording a very direct and concise representation of the concurrent system being modeled.

Maude is also a high-performance language. For example, in a recent detailed benchmarking of 15 well-known algebraic, functional and object-oriented languages by Hubert Garabel and his collaborators at INRIA Rhône-Alpes, Haskell and Maude were the two languages showing the highest overall performance [19].

For the purposes of this paper, the main points to emphasize are that, once a distributed system design has been expressed in Maude as a rewrite theory:

- Such a system design can be *simulated* using Maude's `frewrite` command.
- Its reachability properties, including both failures of invariants and monitoring of consistency and other properties can be exhaustively analyzed by breadth first search using Maude's `search` command.
- Provided that the set of states reachable from a given initial state is finite, its LTL temporal logic properties can be analyzed using Maude's LTL model checker [15].
- To specify the system's real-time aspects and model check its real-time temporal logic properties the Real-Time Maude language and system can be used [35].
- Expressing the system's probabilisitic aspects as a *probabilistic rewrite theory* [1], its quantitative performance aspects can be analyzed by statistical model checking using the PVeStA tool [2].

In all these ways, Maude supports a style of formal specification and analysis of cloud computing systems that effectively meets the challenge to formal methods, explained in Section 1, that such systems pose. In fact, to the best of my knowledge it seems fair to say that rewriting logic as supported by Maude is the first formally based approach in which both correctness and performance aspects of cloud computing systems have been systematically analyzed.

3 Specifying and Analyzing Cloud Computing Systems

The work on specifying and analyzing cloud computing systems in Maude has focused on two main areas: (i) formal specification and analysis of cloud storage systems; and (ii) some security aspects of cloud computing.

3.1 Formal Specification and Analysis of Cloud Storage Systems

Only a short summary of work in this area is possible here. I refer to the survey [12], from which the summary of the work on systems (1)–(3) is drawn, for a detailed account. My summary of (4) is based on [36], and that of (5)–(6) on [31,30]. The cloud storage systems that have been specified and analyzed in Maude include:

1. **Apache Cassandra** [23] is an open-source industrial key-value data store having about 345,000 lines of code that only guarantees *eventual consistency*. To the best of our knowledge, before our work no formal specification of Cassandra existed and, although believed to guarantee eventual consistency, no verification of that property had been carried out. After studying Cassandra's code, we first developed a 1,000-line Maude specification with just 20 rewrite rules [32] that captured the system's main components, such as data partitioning strategies, consistency levels, and timestamp policies for ordering multiple versions of data. Standard model checking allowed us to confirm that Cassandra does support eventual consistency and to analyze under what conditions Cassandra can guarantee strong consistency. To also analyze Cassandra's performance features and those of a design alternative, we then develop a probabilistic model of Cassandra in [27]. By modifying a single function in our Maude model we obtained a model of our proposed design alternative. The statistical model checking analysis of the original Cassandra model and our alternative Cassandra-like design in PVeStA indicated that the proposed design alternative did *not* improve Cassandra's performance. But this left open the question of how reliable these analyses were. To answer this question we modified the Cassandra code to obtain an implementation of the alternative design, and executed both the original Cassandra code and the new system on representative workloads. These experiments showed that PVeStA statistical model checking provides reliable performance estimates.

2. **Megastore** [11] is a key part of Google's cloud infrastructure. Megastore's trade-off between consistency and efficiency is to guarantee consistency only for transactions that access a single *entity group* (e.g., "John's email" or "books on formal verification"). Megastore's code is not publicly available, and only a short high-level description has been given in [11]. To fully understand the Megastore algorithms Jon Grov and Peter Ölveczky first developed in [20] a sufficiently detailed executable formal specification of Megastore in Real-Time Maude based on the description in [11]. This is the first publicly available formalization and reasonably detailed description of Megastore. It contains 56 rewrite rules, of which 37 deal with fault tolerance features.

   To analyze both the correctness and the performance of Megastore's Maude model, two additional models were developed: (i) since in the original real-time model only those behaviors that are possible *within the given timing parameters* are analyzed, to exhaustively analyze all possible system behaviors *irrespective* of particular timing parameters, an *untimed* model was also developed; and (ii) for *performance estimation* purposes, a real-time model in which certain parameters, such as the messaging delays between two nodes, are selected probabilistically according to a given probability distribution was also developed.

   Furthermore, Jon Grov had an idea on how to extend Megastore so that it would also guarantee strong consistency for certain transactions accessing *multiple* entity groups *without* sacrificing performance. This led to the design

of *Megastore-CGC*. The key observation is that a Megastore site replicating a set of entity groups participates in all updates of these entity groups and should therefore be able to maintain an ordering on those updates. The idea behind the Megastore-CGC extension is that, by making this ordering explicit, such an "ordering site" can validate transactions [21]. Since Megastore-CGC exploits the implicit ordering of updates during Megastore commits, it *piggybacks* ordering and validation onto Megastore's commit protocol and therefore does not require additional messages for validation and commit. A *failover* protocol deals with failures of the ordering sites. Both simulations (to discover performance bottlenecks) and Maude model checking were extensively used during the development of Megastore-CGC, whose formalization contains 72 rewrite rules. The performance estimated for Megastore and Megastore-CGC using randomized simulations in Real-Time Maude indicated that both system designs had about the same performance. That is, a design with considerably stronger consistency guarantees was obtained without sacrificing performance.

3. **RAMP.** Read-Atomic Multi-Partition (RAMP) transactions were proposed by Peter Bailis *et al.* [10] to offer light-weight multi-partition transactions that guarantee one of the fundamental consistency levels, namely, *read atomicity*: either all updates or no updates of a transaction are visible to other transactions. The paper [10] gives hand proofs of correctness properties and proposes a number of variations of RAMP without giving details. We used Maude to: (i) check whether RAMP indeed satisfies the guaranteed properties, and (ii) develop detailed specifications of the different variations of RAMP and check which properties they satisfy. Specifically, in [29,26] we used reachability analysis to analyze whether the different variants of RAMP satisfy the following properties (from [10]):
   - *Read atomic isolation*: either all updates or no updates of a transaction are visible to other transactions.
   - *Companions present*: if a version is committed, then each of the version's sibling versions are present on their respective partitions.
   - *Synchronization independence*: each transaction will eventually commit or abort.
   - *Read your writes*: a client's writes are visible to her subsequent reads.

   We analyzed these properties for our seven versions of RAMP. Our analysis results agree with the theorems and conjectures in [10]: all versions satisfy the above properties, except that: (i) RAMP without 2PC only satisfies synchronization independence; and (ii) RAMP with one-phase writes does not satisfy read-your-writes.

   Furthermore, in [28] we used statistical model checking to analyze whether the different variants of RAMP offer the expected performance (only two of the versions were implemented by the RAMP developers for performance analysis). Our statistical model checking performance results: (a) were consistent with the experimental evaluations of the two implemented designs; (b) were also consistent with conjectures made by the RAMP developers for

other unimplemented designs; and (c) have uncovered some promising new designs that seem attractive for some applications.

4. **P-Store** [39] P-Store is a data store that combines wide-area replication, data partition, some fault tolerance, serializability, and limited use of atomic multicast. It has influenced other recent data store designs that can be seen as extensions of its design. P-Store uses atomic multicast to order concurrent transactions and group communication for atomic commit. As pointed out for example in [5], both atomic multicast and group communication commit seem to be key building blocks in cloud storage systems. However such features were not formalized in previous work. Indeed, Ölveczky's paper on P-Store [36] describes the formalization and formal analysis of P-Store in Maude and, as part of its main contributions, specifies group communication commitment, and defines an abstract Maude model of atomic multicast that allows any possible ordering of message reception consistent with atomic multicast. Besides providing a Maude formal model of two versions of P-Store, the work in [36] performed model checking analysis. This analysis uncovered some significant errors in the supposedly-verified P-Store algorithm, like read-only transactions never getting validated in certain cases. One of the authors of the original P-Store paper [39] did confirm that a nontrivial mistake had been found in their algorithm and suggested a way of correcting the mistake. The Maude analysis of the corrected algorithm did not find any errors. Furthermore, the analysis showed that a crucial assumption was missing from the original P-Store paper, and that a key definition was very easy to misunderstand because of how it was phrased in English. All this showed that there is a clear need for formal specification and analysis beyond the standard prose-cum-pseudo-code descriptions and informal correctness proofs.
5. **Walter** [42] is a distributed partially replicated data store providing Parallel Snapshot Isolation (PSI), an important consistency property that offers attractive performance while ensuring adequate guarantees for certain kinds of applications. Walter is a very good opportunity for formal methods, because no formal system specification existed at all before our work in [31], and there was no formal (or even informal) verification that it guarantees PSI. Furthermore, Walter is also a good stepping stone towards placing the design of cloud-based transaction systems in a formally-based modular framework. In this sense, Walter has been a key missing design in the spectrum, so that its study complements and enriches the general picture that has been obtained in the formal modeling and analysis studies on Cassandra, Megastore, RAMP, P-Store, and ROLA discussed above in (1)–(4) and below on (6).
   In [31] we have:
   - Given in Maude the first formal executable specification of Walter.
   - Formalized the SI and PSI properties and formally analyzed for the first time whether the Walter design satisfies either of these properties. This analysis has been achieved by: (a) providing a parametric method to generate all initial states for given parameters; and (b) performing model

checking analysis to verify the SI and PSI properties for all initial states for various parameter choices. Our analysis shows that the Walter design does indeed satisfy the PSI property for all our initial states but fails to satisfy the SI property;
- Extended the Maude model of Walter from a rewrite theory to a probabilistic rewrite theory by adding time and probability distributions for message delays to the original specification. We then carried out a systematic statistical model checking analysis of the key performance metric, transaction throughput, under a wide range of workloads. The results of this analysis confirms that the performance estimates thus obtained are consistent with those obtained experimentally for the Walter implementation in [42]; and they furthermore provide new insights about Walter's performance beyond the limited ranges for which such information was available by experimental evaluation in [42].

6. **ROLA** [30] is a new distributed transaction protocol that has been designed and analyzed using Maude from the very beginning. Different applications require negotiating the consistency vs. performance trade-offs in different ways. The point of ROLA is to explore a specific such tradeoff not studied before. The key issue is the required degree of consistency for a given application, and how to meet its consistency requirements with high performance. Cerone et al. [14] survey a hierarchy of consistency models for distributed transaction protocols including (in increasing order of strength): (i) read atomicity (RA): either all or none of a distributed transactions updates are visible to another transaction (that is, there are no fractured reads); (ii) causal consistency (CC): if transaction T2 is causally dependent on transaction T1, then if another transaction sees the updates by T2, it must also see the updates of T1 (e.g., if A posts something on a social media, and C sees Bs comment on As post, then C must also see As original post); (iii) parallel snapshot isolation (PSI): like CC but without lost updates; and so on, all the way up to the well-known serializability guarantees. A key property of transaction protocols is the prevention of lost updates (PLU). The weakest consistency model in [14] satisfying both RA and PLU is PSI. However, PSI, and the already discussed Walter protocol [42] implementing PSI, also guarantee CC. Cerone et al. conjecture that a system guaranteeing RA and PLU without guaranteeing CC should be useful, but up to now we are not aware of any such protocol. The point of ROLA is exactly to fill this gap: guaranteeing RA and PLU, but not CC. Two key questions are then: (a) are there applications needing high performance where RA plus PLU provide a sufficient degree of consistency? and (b) can a new design meeting RA plus PLU outperform existing designs, like Walter, meeting PSI?

   Regarding question (a), an example of a transaction that requires RA and PLU but not CC is the becoming friends transaction on social media. Bailis et al. [9] point out that RA is crucial for this operation: If Edinson and Neymar become friends, then Unai should not see a fractured read where Edinson is a friend of Neymar, but Neymar is not a friend of Edinson. An

implementation of becoming friends must obviously guarantee PLU: the new friendship between Edinson and Neymar should not be lost. Finally, CC could be sacrificed for the sake of performance: Assume that Dani is a friend of Neymar. When Edinson becomes Neymar's friend, he sees that Dani is Neymar's friend, and therefore also becomes a friend of Dani. The second friendship therefore causally depends on the first one. However, it does not seem crucial that others are aware of this causality: If Unai sees that Edinson and Dani are friends, then it is not necessary that he knows that (this happened because) Edinson and Neymar are friends.
Regarding question (b), the work in [30] compared the performance of ROLA with that of Walter. To model time and performance issues, ROLA has been specified in Maude as a probabilistic rewrite theory. ROLAs RA and PLU requirements were then analyzed by standard model checking disregarding time issues. To estimate ROLAs performance, and to compare it with that of Walter, the specification of Walter in Maude was used, and the Maude models of both ROLA and Walter were analyzed by statistical model checking analysis using the PVeStA tool. The results of this analysis showed that ROLA outperforms Walter in all performance requirements for all read/write transaction rates. To the best of our knowledge this is the first demonstration that, by a suitable use of formal methods, a completely new distributed transaction protocol can be designed and thoroughly analyzed, as well as be compared with other designs, very early on, before its implementation.

3.2 Some Security Aspects of Cloud Computing Systems

The work on using formal specification and analysis in Maude for cloud computing security is less developed than that on storage systems, but it can give a taste for what is possible. A common theme running through both of the studies that I summarize below is that cloud computing, while giving rise to new security vulnerabilities, does also offer the possibility of arriving at system designs that take advantage of cloud computing to increase system security. My summary of (1) is based on material in [16], and that of (2) on [40,12]

1. **Achieving Stable Availability in the Face of DDoS Attacks**. Availability is a crucial security property for cloud-based systems. It can be compromised by distributed Denial of Service (DDoS) attacks. In [16] two Maude-based formal patterns (in the sense of [33]), and their combination into the ASV+SR pattern were presented. Used in their ASV+SR combination, they can effectively defend cloud-based systems against DDoS attacks. The key notion proposed is that of *stable availability*, meaning that, with very high probability, service quality remains very close to a chosen threshold, regardless of how bad the DDoS attack can get. This notion is a good example of how cloud computing can be used to enhance security, in this case defenses against DDoS attacks. The two most basic formal patterns used as defenses against DDoS attacks were: (i) the Adaptive Selective Verification (ASV) pattern, which enhances a communication protocol with a

defense mechanism, and (ii) the Server Replicator (SR) pattern, which exploits cloud computing's flexibility to provision additional resources based on perceived congestion. However, ASV achieves availability without stability, and SR cannot achieve stable availability at a reasonable cost. As a main result the work in [16] shows, by statistical model checking with the PVeStA tool, that (iii) the ASV+SR composition of both patterns yields a new pattern which guarantees stable availability at a reasonable cost.

The key problem addressed is that DDoS defense mechanisms that help maintaining availability can nevertheless show performance degradation as a DDoS attack worsens. Thus, a key goal in [16] is to design DDoS security adaptive measures that can achieve stable availability, which means that with very high probability service quality remains very close to a chosen constant quantity, which does not change over time, regardless of how bad the DDoS attack can get. Cloud Computing, by offering the possibility of dynamic resource allocation, can be used to leverage stable availability when combined with DoS defense mechanisms.

The ASV protocol [25,3] is a well-known defense against DDoS attacks in the typical situation that clients and attackers use a shared channel where neither the attacker nor the client have full control over the communication channel [25]. The ASV protocol adapts to increasingly severe DDoS attacks and provides improved availability. However, it cannot provide stable availability. By replicating servers one can dynamically provision more resources to adapt to high demand situations and achieve stable availability; but the cost of provisioned servers drastically increases in a DDoS attack situation. These two patterns are modeled in Maude and then formally composed to obtain the new ASV+SR pattern. As a main result the work in [16] shows, by analyzing the quantitative properties of ASV+SR with the statistical model checker PVeStA, that ASV+SR guarantees stable availability at a reasonable cost. The key idea of ASV+SR is relatively easy to explain. As a DDoS attack gets worse, ASV servers randomly drop an increasing number of messages from clients, and honest clients increase their resending of messages based on their perceived latency to get a server's response. ASV ensures that messages from honest clients *will* eventually get through with very high probability; but performance is degraded. The ASV+SR protocol avoids this performance degradation. However, *much fewer additional servers* need to be provisioned than if a naive approach based only on SR were used. Actually, in ASV+SR the threshold for provisioning new servers is itself a chosen parameter: one can settle for a small, constant factor in performance degradation at the expense of substantial savings in the provisioning of new servers.

2. **Building a Group Key Management Service on top of ZooKeeper**. Zookeper [24] is a fault-tolerant distributed key/value data store that provides reliable distributed coordination. The work in [40] investigated whether a useful group key management service can be built using ZooKeeper using Maude and statistical model checking in PVeStA.

Group key management is the management of cryptographic keys for secure communication between multiple authorized entities. A central group key controller can fulfill this need by: (a) authenticating/admitting authorized users into the group, and (b) generating a *group key* and distributing it to authorized group members [43]. In settings with a centralized group controller, its failure can impact both group dynamics and periodic key updates, leaving the group key vulnerable. This is especially significant when designing a cloud-based group key management service, since such a service will likely manage many groups.

The work in [40] investigated whether a fault-tolerant cloud-based group key management service could be built by leveraging existing coordination services commonly available in cloud infrastructures and if so, how to design such a system. In particular, we: (a) designed a group key management service built using Zookeeper [24], a reliable distributed coordination service supporting Internet-scale distributed applications, (b) developed a rewriting logic model of our design in Maude [15], based on [22], where key generation is handled by a centralized key management server and key distribution is offloaded to a ZooKeeper cluster and where the group controller stores its state in ZooKeeper to enable quick recovery from failure, and (c) analyzed our model using the PVeStA [4] statistical model checking tool. The analysis centered on two key questions: (1) can a ZooKeeper-based group key management service handle faults more reliably than a traditional centralized group key manager, and (2) can it scale to a large number of concurrent clients with a low enough latency to be useful?

Our analysis consisted of two experiments. Both were run hundreds of times via PVeStA and average results were collected. The first experiment was designed to test whether saving snapshots of the group key manager's state in the ZooKeeper store could increase the overall reliability of the system.

In the first experiment we compared the average key manager availability (i.e., the time it is available to distribute keys to clients) between a single key manager and two key managers where they share a common state saved in the ZooKeeper store. We observed an availability improvement from 65% to 85%. Our second experiment was designed to examine whether using ZooKeeper to distribute shared keys is efficient and scalable enough for real-world use. The experiment measured the variations in: (a) the percentage of keys successfully received by group members, and (b) the key distribution latency, as increasing numbers of clients joined a group per second. We analyzed our original model and a slightly modified model where we added a 2 second wait time between key updates from the key manager. While our initial experiments show that naively using ZooKeeper as a key distribution agent works well, at high client join rates, the key reception rate leveled out around 96%. This occurs because ZooKeeper can apply key updates internally more quickly then clients can download them. By adding extra latency between key updates, the ZooKeeper servers are forced to wait enough time for the correct keys to propagate to clients, the slightly modified design achieved a 99% key reception in all cases. On the other hand, key

distribution latency remained relatively constant, at around half a second, regardless of the join rate because ZooKeeper can distribute keys at a much higher rate than a key manager can update them [24].
In essence, our analysis confirmed that a scalable and fault-tolerant key-management service can indeed be built using ZooKeeper, settling various doubts raised about the effectiveness of ZooKeeper for key management by an earlier, but considerably less-detailed, model and analysis [17]. This result is not particularly surprising, especially considering that many man-hours would be needed to optimize an actual system. More interestingly, the analysis also showed that system designs may suffer from performance bottlenecks not readily apparent in the original description—highlighting the power of formal modeling and analysis as a method to explore the design space.

4 Limitations and Some Future Directions

One important limitation of this extended abstract is that there is no room for a careful comparison with related work. Fortunately, a quite up to date such comparison has been given in the survey [12], to which I refer for a discussion of other work in this area. Two other current limitations pointing to future research directions are: (1) the absence at the moment of full verification by theorem proving for the systems that I have discussed; and (2) the current status of Maude executable specifications as *prototypes* useful for simulation and analysis, but not used for the moment for distributed implementations.

Regarding limitation (1), the obvious thing to say is that theorem proving is a natural next step. I have emphasized earlier —and the various systems I have discussed have further stressed— that perhaps the first and most valuable service that Maude executable specifications can render to cloud computing is not verification per se, but rather *fast design exploration guided by formal analysis*. It makes no sense to model check the *wrong* design. And, due to the labor intensive nature of theorem proving, it makes even less sense to perform theorem proving verification on such a wrong design, particularly since theorem provers are not that good at finding counterexamples and, furthermore, in this area logical correctness is only part of the story: performance matters quite as much. Theorem proving is, as I said, a complementary next step: after having arrived at a good system design and having thoroughly analyzed its logical correctness properties —resp. its performance— by standard model checking —resp. by statistical model checking— for representative initial states, the next step is to fully verify the systems key logical properties for *all initial states* by theorem proving. For Maude specifications of distributed systems, three related approaches, one based on symbolic model checking and two based on theorem proving, seem particularly well suited:

– The *Logical Model Checking* approach in [18,6,7,8] is in some sense halfway between model checking and theorem proving: it allows full verification of temporal logic properties for *infinite-state systems* and for *infinite sets of initial states*.

- The deductive verification of invariants and other safety properties by the unification methods supported by Maude's *Invariant Analyzer* tool [37,38] is also directly relevant and can be a useful tool for verifying invariants.
- The *Constructor-Based Reachability Logic* for rewrite theories presented in [41] is a third attractive alternative. Reachability logic generalizes Hoare logic and can express many Hoare-like partial correctness properties, including invariants. Although its tool is still under development, it has already been applied to the deductive verification of some distributed systems.

Regarding limitation (2) there are two main things to say. First, thanks to Maude's support for TCP/IP sockets as built-in objects [15], Maude programs can be easily distributed. The basic idea is that objects in a distributed system written in Maude can be executed in different machines, with sockets used to perform message passing communication across machines. What is needed, however, is to make the passage from a Maude model to its distributed implementation as simple and as efficient as possible. Current, as yet unpublished, research is advancing this direction. In particular, distributed storage systems are among the examples we are experimenting with. Second, this direction is particularly important to arrive at system implementations that are *correct by construction*. In fact, this dovetails very nicely with the effort in overcoming limitation (1), since all this should make it possible to generate correct by construction distributed implementations from Maude-based formal specifications of system designs that have already been submitted to both model checking and theorem proving verification.

Acknowledgments. As the references make clear, most of these ideas have been developed in joint work with a large number of collaborators and former or present students, including: Musab AlTurki, Rakesh Bobba, Jonas Eckhardt, Jatin Ganhotra, Jon Grov, Indranil Gupta, Si Liu, Tobias Mühlbauer, Son Nguyen, Peter Csaba Ölveczky, Muntasir Raihan Rahman, Stephen Skeirik, and Martin Wirsing. Furthermore, in some of the work I report on, such as [20,21,36], I have not been involved. These projects were partially supported by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084, the National Science Foundation under Grant Nos. NSF CNS 1409416 and NSF CNS 1319527, and the Naval Research Laboratory under contract number NRL N00173-17-1-G002. I thank the organizers of SBMF 2018 for giving me the opportunity of presenting these ideas at the meeting in Salvador.

References

1. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electr. Notes Theor. Comput. Sci. 153(2), 213–239 (2006)
2. AlTurki, M., Meseguer, J.: PVeStA: A parallel statistical model-checking and quantitative analysis tool (2011), in Proc. CALCO 2011, Springer LNCS 6859, 386–392

3. AlTurki, M., Meseguer, J., Gunter, C.: Probabilistic modeling and analysis of DoS protection for the ASV protocol. Electr. Notes Theor. Comput. Sci. 234, 3–18 (2009)
4. AlTurki, M., Meseguer, J.: PVeStA: A parallel statistical model checking and quantitative analysis tool. In: Proc. CALCO'11, LNCS, vol. 6859. Springer (2011)
5. Ardekani, M.S., Sutra, P., Shapiro, M.: G-DUR: a middleware for assembling, analyzing, and improving transactional protocols. In: Proceedings of the 15th International Middleware Conference. pp. 13–24. ACM (2014)
6. Bae, K., Escobar, S., Meseguer, J.: Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. In: Rewriting Techniques and Applications (RTA'13). LIPIcs, vol. 21, pp. 81–96. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2013)
7. Bae, K., Meseguer, J.: Infinite-state model checking of LTLR formulas using narrowing. In: Proc. WRLA 2014. LNCS, vol. 8663, pp. 113–129. Springer (2014)
8. Bae, K., Meseguer, J.: Predicate abstraction of rewrite theories. In: RTA-TLCA. Lecture Notes in Computer Science, vol. 8560, pp. 61–76. Springer (2014)
9. Bailis, P., Fekete, A., Ghodsi, A., Hellerstein, J.M., Stoica, I.: Scalable atomic visibility with RAMP transactions. ACM Trans. Database Syst. 41(3), 15:1–15:45 (2016)
10. Bailis, P., Fekete, A., Hellerstein, J.M., Ghodsi, A., Stoica, I.: Scalable atomic visibility with RAMP transactions. In: Proc. SIGMOD'14. ACM (2014)
11. Baker, J., et al.: Megastore: Providing scalable, highly available storage for interactive services. In: CIDR'11. www.cidrdb.org (2011)
12. Bobba, R., Grov, J., Gupta, I., Liu, S., Meseguer, J., Ölveczky, P., Skeirik, S.: Design, formal modeling, and validation of cloud storage systems using Maude. Tech. Rep. `http://hdl.handle.net/2142/96274`, University of Illinois Computer Science Department (June 2017), to appear in R.H. Campbell et al. (eds.) *Assured Cloud Computing*, J. Wiley, 2018
13. Brewer, E.A.: Towards robust distributed systems (abstract). In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. p. 7. ACM (2000)
14. Cerone, A., Bernardi, G., Gotsman, A.: A framework for transactional consistency models with atomic visibility. In: Proc. 26th International Conference on Concurrency Theory, CONCUR 2015. LIPIcs, vol. 42, pp. 58–71. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
15. Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P., Martí-Oliet, N., Talcott, C.: All About Maude – A High-Performance Logical Framework. Springer LNCS Vol. 4350 (2007)
16. Eckhardt, J., Mühlbauer, T., AlTurki, M., Meseguer, J., Wirsing, M.: Stable availability under denial of service attacks through formal patterns. In: de Lara, Zisman (eds.) FASE. LNCS, vol. 7212, pp. 78–93. Springer (2012)
17. Eckhart, J.: Security Analysis in Cloud Computing using Rewriting Logic. Master's thesis, Ludwig-Maximilans-Universität München (2012)
18. Escobar, S., Meseguer, J.: Symbolic model checking of infinite-state systems using narrowing. In: Proc. RTA. Lecture Notes in Computer Science, vol. 4533, pp. 153–168 (2007)
19. Garavel, H., Tabikh, M., Arrada, I.: Benchmarking implementations of term rewriting and pattern matching in algebraic, functional, and object-oriented languages - the 4th rewrite engines competition. In: Rusu, V. (ed.) Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018. Lecture Notes in Computer Science, vol. 11152, pp. 1–25. Springer (2018)

20. Grov, J., Ölveczky, P.C.: Formal modeling and analysis of Google's Megastore in Real-Time Maude. In: Specification, Algebra, and Software. LNCS, vol. 8373. Springer (2014)
21. Grov, J., Ölveczky, P.C.: Increasing consistency in multi-site data stores: Megastore-CGC and its formal analysis. In: Proc. SEFM'14. LNCS, vol. 8702. Springer (2014)
22. Gupta, J.: Available group key management for NASPInet. Master's thesis, Univeristy of Illinois at Champaign-Urbana (2011)
23. Hewitt, E.: Cassandra: The Definitive Guide. O'Reilly Media (2010)
24. Hunt, P., Konar, M., Junqueira, F., Reed, B.: Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX ATC. vol. 10 (2010)
25. Khanna, S., Venkatesh, S.S., Fatemieh, O., Khan, F., Gunter, C.A.: Adaptive selective verification. In: INFOCOM. pp. 529–537. IEEE (2008)
26. Liu, S., Ganhotra, J., Rahman, M.R., Nguyen, S., Gupta, I., Meseguer, J.: Quantitative analysis of consistency in NoSQL key-value stores. LITES 4(1), 03:1–03:26 (2017)
27. Liu, S., Nguyen, S., Ganhotra, J., Rahman, M.R., Gupta, I., Meseguer, J.: Quantitative analysis of consistency in NoSQL key-value stores. In: Proc. QEST'15. LNCS, vol. 9259. Springer (2015)
28. Liu, S., Ölveczky, P.C., Ganhotra, J., Gupta, I., Meseguer, J.: Exploring design alternatives for RAMP transactions through statistical model checking. In: Proc. Formal Methods and Software Engineering - 19th International Conference on Formal Engineering Methods, ICFEM 2017. Lecture Notes in Computer Science, vol. 10610, pp. 298–314. Springer (2017)
29. Liu, S., Ölveczky, P.C., Rahman, M.R., Ganhotra, J., Gupta, I., Meseguer, J.: Formal modeling and analysis of RAMP transaction systems. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016. pp. 1700–1707. ACM (2016)
30. Liu, S., Ölveczky, P.C., Santhanam, K., Wang, Q., Gupta, I., Meseguer, J.: ROLA: A new distributed transaction protocol and its formal analysis. In: Proc. Fundamental Approaches to Software Engineering, FASE 2018. Lecture Notes in Computer Science, vol. 10802, pp. 77–93. Springer (2018)
31. Liu, S., Ölveczky, P.C., Wang, Q., Meseguer, J.: Formal modeling and analysis of the Walter transactional data store. In: Rusu, V. (ed.) Proc. Rewriting Logic and Its Applications - 12th International Workshop, WRLA 2018. Lecture Notes in Computer Science, vol. 11152, pp. 136–152. Springer (2018)
32. Liu, S., Rahman, M.R., Skeirik, S., Gupta, I., Meseguer, J.: Formal modeling and analysis of Cassandra in Maude. In: Proc. ICFEM'14. LNCS, vol. 8829. Springer (2014)
33. Meseguer, J.: Taming distributed system complexity through formal patterns. Sci. Comput. Program. 83, 3–34 (2014)
34. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: How Amazon Web Services uses formal methods. Communications of the ACM 58(4), 66–73 (April 2015)
35. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1–2), 161–196 (2007)
36. Ölveczky, P.C.: Formalizing and validating the P-Store replicated data store in Maude. In: Recent Trends in Algebraic Development Techniques - 23rd IFIP WG 1.3 International Workshop, WADT 2016. Lecture Notes in Computer Science, vol. 10644, pp. 189–207. Springer (2016)

37. Rocha, C., Meseguer, J.: Proving safety properties of rewrite theories (2011), in Proc. CALCO 2011, Springer LNCS 6859, 314-328
38. Rocha, C., Meseguer, J.: Mechanical analysis of reliable communication in the alternating bit protocol using the Maude invariant analyzer tool. In: Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi. Lecture Notes in Computer Science, vol. 8373, pp. 603–629. Springer (2014)
39. Schiper, N., Sutra, P., Pedone, F.: P-Store: Genuine partial replication in wide area networks. In: Proc 29th IEEE Symposium on Reliable Distributed Systems (SRDS 2010). pp. 214–224. IEEE Computer Society (2010)
40. Skeirik, S., Bobba, R.B., Meseguer, J.: Formal analysis of fault-tolerant group key management using ZooKeeper. In: 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'13). IEEE Computer Society (2013)
41. Skeirik, S., Stefanescu, A., Meseguer, J.: A constructor-based reachability logic for rewrite theories. In: Proc. Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017. Lecture Notes in Computer Science, vol. 10855, pp. 201–217. Springer (2017)
42. Sovran, Y., Power, R., Aguilera, M.K., Li, J.: Transactional storage for geo-replicated systems. In: Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011. pp. 385–400. ACM (2011)
43. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure group communications using key graphs. IEEE/ACM Trans. Netw. 8(1), 16–30 (2000)