Università
della
Svizzera
italiana

Faculty
of Informatics

Research Prospectus

July 16, 2014

Data Consistency Something

Nosheen Zaza

Abstract

The utilization of relaxed data consistency models has increased in recent years, especially in distributed applications, where maintaining strong consistency come on the cost of up-time and performance. Many systems are deployed successfully and reliably on top of eventually-consistent frameworks, such as cassandra, hadoop, mongoBD..etc Developing systems with correct consistency properties is difficult, as current programming languages do not provide abstrations that enable programmers express their intentions intuitivly and globally. In this research prospectus, we overview techniques and tools, both classic and state of art that cover suitable data abstractions to express consistency, defining consistency levels, and motivate the opprotunity for further research.

Research Advisor
Prof. Nate Nystrom

Academic Advisor
Prof. Nate Nystrom

Review Committee
Prof. Committee Member1, Prof. Committee Member2

Research Advisor's approval (Prof. Nate Nystrom): Date: .....

PhD Director's approval (Prof. Stefan Wolf): Date: .....

1 Introduction and Problem Domain

Relaxed data consistency models are being successfully employed in frameworks on which many types of applications can be built. The internet has brought many use-cases where consistency can be relaxed to a great extent. As a motivating example, consider a shopping website that replicates product data. When a client requests a list of all products that have 5-start ratings, it is not necessary to query all replicas in order to get the most up-to-date rating data. In fact, this is not even recommended, as it would cause unnecessary system load, which triggers unwanted delays for a task that is inherently subjective. Real-world use cases include metrics collection and product recommendations at eBay [**?**], and inbox search at facebook. [4]. Both employ an eventually-consistent [5] storage backend.

Relaxed consistency is rarely sufficient on its own for the requirements of a complete application. It is often the case that components requiring different consistency policies co-exist and operate. Great care must be taken to ensure correct program semantics in such cases. Continuing from the previous example, consider the case when a client wants to buy an item from the list of items with 5-star ratings. Because the list was not constructed by querying all replicas, it is possible that it includes some items that are no longer in stock. If the programmer forgets to perform query all replicas, to check that the selected item is in stock prior to check out. The client will end up buying a non-existent item. Figure 1 depicts this bad scenario.

The problem of maintaining data consistency is not exclusive to distributed systems with explicit replication. It is well known and documented in in concurrent programming literature. On the lowest level possible, data is replicated under the hood among main memory, caches and registers, and maintaining data consistency become the responsibility of the programmer, who must ensure that

1. His program does not violate consistency.

2. The compiler will not reoder his program in a way that violates consistency.

3. The CPU will not reorder his program instruction in a way that violates conszistency.

Many techniques designed to ensure these do not happen. Some can be successfully adapted to a distributed environment. The fact that replication is explicit makes it even a better environment to implement many techniques, and this is a topic for active research.

In this research prospectus, we overview

Unlike distributed applications, multicore applications typically require strong consistency on the application level, even if the underlying model does not provide it.

The mechanisms cannot delay much, so some techniques that might not be suitable for networked applicatiosn absolutely do not work there, such as transactions. There have been attemps to implement transactional memory, but performance is usually the biggest obstacle.

On the other hand, there are issues special to distributed systems that also make maintaining consistency hard. You usually do not have to consider, in your multicore application, one core dying down and losing data, or your instructions or data getting lost between wires, these issues are the first things to be thought about when designing a distributed application. packet loss, and servers going down problems are not strange to distributed systems designers.

Cases where information among replicas are not the same are less common that one might think, typically, it would take milliseconds to synchronize information at all replicas [2], however, with some systems processing millions of requests, rare events do occur, and must be considered when designing a system [5]. Also, there are cases where we want to ensure not only coherence of information among all replicas, but we also need ordered access.

We want to answer the following: How should a programming language be designed to support various levels of application level data consistency in a distributed environment? Throughout programming history, designing programming languages was about detecting common coding practices and coming up with constructs that would encode them more efficiently, as well as protecting the programmers from making common programming mistakes. Java bakes in object locks, the infamous synchronized keyword, as well as volatile. Adding checked exceptions forces the programmer to handle them. Futures in Scala force also enforce handling error by default rather than leaving it as a forgettable option, and also manitest the fact that something will take time, and thus this should be handled. What else can we bake in to support better syntax and protection?

Why Distributed environments? Beause there is many applications of “lies, damned lies and statistics” [6] on the application level. No hidden replication, and the environment in general is cleaner.

1.

2 Properties of Distributed Systems

Distributed systems exhibit properties, or can benefit from lock-free, wait free algorithms. For example a recent addition to the Apache Cassandra [3] storayge is light-weight transactions, which are not transactions as used in transactional memory and databases

A data consistency model is needed whenever multiple copies of data often replicate (note that replication is not the only source of consistency problems) [1], and access synchronization. Often, underlying systems do not provide strong consistency guarantees, and ensuring consistency is moved to the application layer, examples are CPU architectures[], or eventually consistent systems. Sometime we also want to allow relaxed consistency to improve performance and increase consistency, such in [skip lists], or many distributed applications [ref].

There has been many tools and methodologies to enable users express and enforce consistency. Most used are locks[ref], transactions[ref], recently low-level hardware instructions are being exposed to programmers[ref]. Problem with locks is that they do not compose, and you could aquire too many or too few of them, with transaction is that their overhead cannot be tolerated in many environments, with low-level instructions is that they are hard to use.

There are many applications to analyse a program and detect races and thus help the programmer to aquire enough locks, transactional memory was being worked on[ref]. Generally these approaches view ensuring consistency as a property of operations, and you should write the operation in a way to ensure required properties of data.

We are interested in another approach with the opposite view, consistency is a property of data, and should be preserved by operations. There are many systems that impose constraints on data to be satisfied, statically or dynamically. [reference some here].

Notable work is atomic sets, where data which we should be accessed serially is marked so, then the compiler takes care and generate the locks. It is still possible to have high-level data races, or just generic data races, if the programmer forgets to merge atomic sets. But this is dependent on the semantics of the program, and computer programs do not understand semantics usually. Anyway attaching consistency to data other than operations has many benefits. the reasoning about data becomes global, there is less redundancy and less room for bugs as operations are ensured to access data in a suitable matter.

This works ok when we want to have serializability vs mess. But what if we have other consistency levels?

There are many problems related to this as well. What are operations in an object-oriented language? What are data boundaries? How to compose effeciently?

Also if we merge two sets, what kind of locks should we aquire for sets with different consistencies?

However the ideas presented there are applicable in distributed environments, there is no data sharing, there are many consistency levels that make sense, and the data containers are less messy than objects.

In this work we focus on distributed data stores, and use as an example the Apache Cassandra store.

Review of such systems and refs.

Operations are clear, reads and writes.

In this work we focus on managing consistency of replicated data in distributed environments.

Want to talk about wait free things and delegated isolation.

Bottom line: having data-centric consistency and replication policies, parameterized by possibly load and what not is interesting, and this is what i plan to work on.

Another bottomer-line: We need to provide better abstractions for data consistency in programming languages and software frameworks.

3 Stats of the Art

In this section, we briefly overview classic and recent research that cover data consistency. Often the research does not necessarily or explicitly focus on consistency, but we will show how we can benefit and employ various aspects to come up with suitable abstractions. The overviews research can be divided as follows

- defining various consistency levels: sql isolation levels yield datasets with certain properties, same for consistency levels for reads and writes in a key-value store, or consistenct definitios for concurrent operations on computer memory.
- Suitable data abstractions: Regions, objects, actors, tables...
- ensuring that operations do not violate data consistency: by showing that programs with certain properties maintain consistency, such as montonic update, share-nothing, separating local and shared data (loci, actors),

showing that programs are deterministic, immutable state. Or with data flow and program analysis techniques. Generally dececting, or enabling the user to express certain data properties that ensure consistency.

- consistency constructs: memory barriers, locks, transactions...
- consistency and how it affects other program properties: can we parameterize programs somwhat to easily tune consistency vs. other propoerties?

3.1 Consistency Levels Definitions

Because different systems have different requirements for consistency, depending on environment parameters (I need to mention some), there are no unified consistency definitions, even though many similarities exist and they overlap. This makes it harder to imbed generic data consistency constructs that would for every application type. Our research direction is towards creating abstractions for distributed systems, because delays can be tolerated, consistency levels are better defined. We however show other consistency definition work.

1.

References

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *computer*, 29(12):66–76, 1996.

[2] A. Cockcroft and D. Sheahan. Benchmarking cassandra scalability on aws-over a million writes per second. *URL: http://techblog. netflix. com/2011/11/benchmarking-cassandra-scalability-on. html (visited on 05/20/2013)*, 2011.

[3] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[4] F. Qu and A. Jambhekar. Cassandra at ebay scale. `http://www.slideshare.net/planetcassandra/5-feng-qu`, 2013.

[5] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[6] Wikipedia. Lies, damned lies, and statistics — wikipedia, the free encyclopedia, 2014. [Online; accessed 14-July-2014].