Optimistic Control for a Critical System

Tom Benedictus Jordi Martori Rune Skou Larsen
Pascal Urso

September 24, 2014

For the last couple of decades building a new IT system has often started with deciding on which Relational Database Management System (RDBMS) to use. A number of factors have proven that this approach comes with drawbacks. It has only been the brave who dared to say that Atomicity, Consistency, Isolation, Durability (ACID) was not always desirable and with Brewer's CAP theorem the debate has flamed around whether you could have consistency, availability, and partition tolerance at the same time. Data now exists in mobile devices, in cloud databases, and in transition on communication lines. This makes partition tolerance a feature of any contemporary solution and availability is a requirement that may overshadow the aim for consistency.

Many recent discussions have been addressing the degree of consistency a large-scale distributed applications and their databases should support. Eventual Consistency (EC) has be promoted as an answer for high-availability systems [5]. Some authors promote higher levels of consistency. For instance, Lloyd et al. [4] describe the issues around building a geo-replicated storage. They promote a consistency model where operation execution may be delayed due to a missing intermediate piece in the communication sequence. We do not argue against the fact that building systems on top of a stronger consistency model – as oppose to building on eventual consistency – is easier or that performance of such systems are well suited to massive-scale applications. But, in our work for designing, implementing and operating real large-scale distributed applications we know that in real system any scenario can occur (due to human error, unpredictable failure, etc...) and we have come across many different use cases including critical systems that cannot suffer any delay or locking. Such a system is FMK, a Shared Medication Record/Medicine Profile of each Danish citizen, where the availability of data is critical. Deliberately this has been built around fully optimistic eventual consistency. It is a clear requirement that the system makes no decisions based on good intentions that could lead to unintended side effects.

Many aspects of the FMK system are managed in an "optimistic" way. All available information must be presented also when not replicated to every data centre even if dependent information is missing or erroneous (e.g. medicine dispensing without the corresponding prescription). Conflicting information must be presented as such and should not be resolved automatically.

1 System Requirements

ACID is often regarded as the classic desirable requirements for a database. As system designers, we completely agree that these requirements help to structure adequately an application built upon a database system, and we want to retain these. Although, we also realise that, in some cases, they must be relaxed in some areas to accommodate real life interactions between humans, devices, servers, communication and storage. Atomicity gives us that either we can carry out the entire transaction or it will be rolled back to the state before it was even started. In a large-scale distributed application this criteria cannot be accommodated as it results in deadlocks. I.e. you cannot reserve all services required for a trip for as long as it has taken you to book them all and retain the right to cancel any as it turns out not to fit in the final possible itinerary. Consistency cannot be guaranteed as part of the atomic transaction as we can't retain the atomic transaction. Isolation in a large-scale distributed application with data producers, consumers and not least the data itself spread over many places and viewed from many perspectives becomes a show stopper in its own right. Airlines will over-book and then handle the potential conflicts subsequently. Amazon may be selling the same last copy of a book in two parallel transactions and provided both sales actually complete they realize that they now have a new issue to solve. Durability may be the one part of ACID that can be accommodated, but even here fact is that we want to get rid of data. Either we are legally not allowed to keep data indefinitely, we are requested to remove a persons data, requested to remove data of a certain nature, or we cannot afford the storage space and processing time involved in retaining data over a longer period of time. When we are getting rid of data we do not like to verify all kind of constraints. Often data in large storage's are disposed in large volumes. This process has to be effective as it is affecting the operational storage.

On the other hand, basically available, soft state, eventual consistency (BASE) [5] one key element in the concept for a contemporary replicated data store. In fact, the data store comprises data traversing the cloud together with data residing on devices, not yet if ever, going to move to another storage. We refer this as data at the edge of the network. We are trusting eventual consistency and provisioning mechanisms for detecting persistent inconsistencies.

In the context of making healthcare decisions, due to possibly lethal interactions between drugs or allergic reaction, it is critical to present all the available information with the highest possible availability. So, it is much better to have some information than none, better to have old information than none, better to have an fragmentary history than a partial but consistent one. Events that happen "outside" the system have indisputably happened, so the system needs to ingest them regardless of the consistency issues. The data itself should not be constrained. E.g. a prescription information with, let's say, a missing/erroneous doctor identification, must not be rejected. Two concurrent updates on the data must be both accessible. All this considerations leads us down the road to a Conflict-free Replicated Data Type (CRDT) [6] data model deployed on Riak

database (dynamo-style eventual consistency database with write/write-conflict capture). The central patient information data model is essentially a state-full CRDT, that exposes a semantic model for write conflicts. Ideally, there would be a replica of the entire service+dataset in each major geographical region/hospital, which is still an eventual goal. Writes should be propagated "as soon as possible", but lack of such propagation - WAN failure - cannot render the system unusable.

2 The FMK system

We are presenting one specific use case from the Health Sector. The system, FMK [1], is holding records of medicine prescribed and administrated to any patient in Denmark and the system is offered in all organisational functions within the Danish health care sector. Hospital, private medical practitioners, pharmacies can access to the system to obtain the whole medical profile of a patient. The patient can also check its own medical record.

On a conceptual layer, this may seem a quite simple online system: for each person, maintain a list of current treatments, which may involve one or more prescriptions, and additionally a set of events that has occurred for the given treatment. Not all treatments require prescriptions. I.e., the doctor can tell you to drink water, or take calcium tablets which you can get without a prescription, but he may make a prescription on these too. Everything prescribed will be in the system. Events are things that happen in the real world, such as a drug being administered to a patient by a nurse, or a drug being handed out at a pharmacy. The system is not an electronic health record with specialised information such as test results, measurements or the like. All that specialised information will be stored in the archives of the hospital or medical facility. The complete patient profile includes :

- electronic prescriptions as well as information about hand-written ones (transcriptions subject to human errors)
- prescribed medicine bought at the pharmacy
- information about the patient's general practitioner, medicine allergies, and reimbursement,
- a check of possible prescribed medicine interaction
- a log to see who has looked up information in patient's medicine profile, including which details have been viewed,

One of the main characteristics of the FMK system is that it has been made to create a single platform to store treatment-prescriptions per patient in all Danish citizens. This provides one of the first successful attempts to create an unified medical platform for this kind of information.

So far the system has not replaced the older, already functioning systems of each hospital, pharmacy and General Practitioner (GP). Nevertheless its been working alongside them providing this unified view of the patients treatments.

One of the collaboration requirements between systems is that, once a patient is admitted in, the hospital staff flags the patient FMK record as been in the hospital and will update the patient information once this patient has been released. So for as long as the patient is in the hospital his/her record will not be consistent with his/her status. This flag, once propagated into the system lets other medical workers know that the information they are reading may not take into account the latest treatments received by the patient. By using this flagging system, the patients information is never blocked which would decrease the availability. Out of the three desirable qualities of a distributed system (Consistency, Availability, and Partition tolerance) the system requires to enforce availability no matter the cost in consistency and the number of failures. So the FMK system only enforces eventual consistency, but only assure that eventually the system will converge and be consistent. However, on the same line, the system only provides a best effort policy, providing the user with the maximum amount of information available, at that current state, thus maximising the reads. Also it will always accept writes and deal later with the merging issues, therefore maximising the writes. Furthermore, all access to any medical record will be logged and both a governmental employee, to check the overall access, and the patient, to check who viewed his information, can check this log.

3 Infrastructure

The wide adoption of this system builds upon a successful cross-sectoral standardisation of medicine workflows and closely related concepts.

One of the primary design criteria for FMK is to provide high availability. The system is in use 24x7, and currently has 40+ integrations with other healthcare systems, most of which are required to use FMK as the primary storage for relevant medicines data.

Though being simple at the high level, much of the challenge lies in making the system highly available, scalable and secure, supporting a wide range of use cases (as well as old APIs), meanwhile making sure that the data that flows in from many of the connected systems has some level of consistency. In many cases data "updates" are made on the basis of a previous "query" to the system, and the system needs to have a model that captures conflicting updates. As such, this seemingly simple system ends up being surprisingly complex, especially because of the high availability requirement.

Figure 1 represents how the main users of the FMK system, i.e. hospitals, pharmacies, GPs and Danish citizens, access the system's information. Pharmacies and GPs use a special application to access the FMK record. Hospitals however access directly through the FMK system to the source, however if the medical staff is not accessing the information from the healthcare infrastruc-

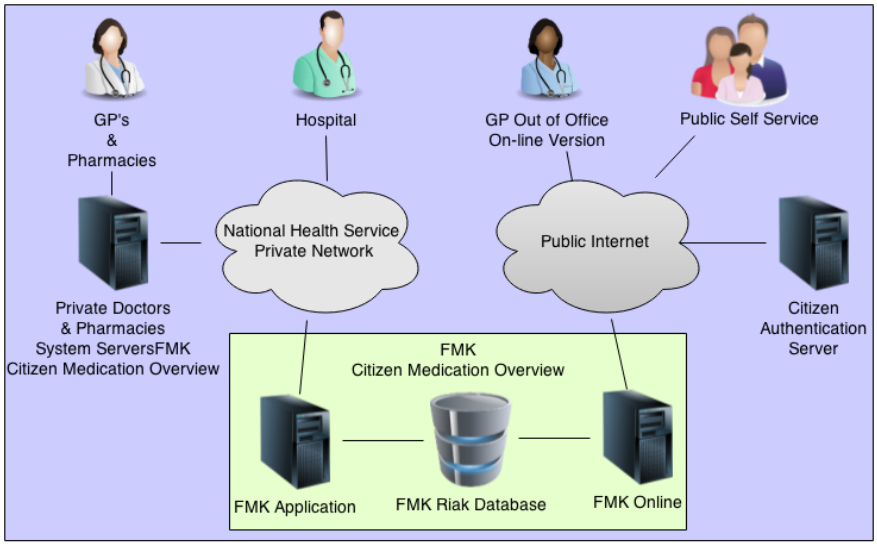

Figure 1: Main FMK users and how the system is accessed.

ture, the National Healthcare Service Private Network, they can still connect to the system authenticating themselves with their unique account. The Danish citizen, in order to check their record and who accessed it, they also do it by connecting to the FMK online with the account.

The system is made up of geographically separated data centres, set up in master-master replication mode, so any Data Centre (DC) can handle any request. The client systems are systems providers for GPs, pharmacies and hospitals as well as a web based system that provides citizen access and acts as a backup for the professional systems. Each client has affinity to a given primary DC, so all requests from a given client use only one DC, as long as it's available.

The FMK consists of a few data centres in Denmark, that work as preferred centres in each respective area, causing the users in each respective area to access mainly their replica unless DC failures are detected or they happen to be temporally closer to the other DC. All application servers in both DC run instances of riak to store the applications information.

Currently the level of integration with the hospitals is fairly low, as such the input of data to the system must be manually done in both systems, the hospital and the FMK applications. That is one of the reasons why when a patient is admitted at the hospital the patient's FMK account is flagged as such, and upon its released the FMK system will be updated with the new treatment and prescriptions, and then unflagged.

As it has been mentioned previously, the main requirement of this system is the availability. In the current system this high availability has been achieved by having several replicas in a few data centres all over Denmark. Each DC

contains a full replica of the system, thus in case of DC failure the other ones will be able to keep functioning.

The system has to deal with all the Danish population, which it is around 5.5 millions of citizens whereas all the basic medical information of each citizen can be add for up to around 100GB. Furthermore, information about the request-log must be stored only during 14 days. Due to Danish legislation after the 14^{th} day all log entries must be forgotten. All this 14-day log occupies also around 100GB. Another log, which is in place, is the audit-log which is also due to Danish laws and has to be stored for a couple of years occupying approximately size of 1TB.

4 Consistency Control

Normally when designing EC systems we focus mainly in the worst case scenario and forget how the system behaves when the amount of errors is tolerable. Considering the system within this tolerable bounds of error allows us to propose two working scenarios.

- Everything goes as it is supposed. Then the system guarantees that we will read-our-writes therefore providing the user with sense of casual consistency.
- In the case of failure, the user maybe able to see outdated information or out of order commit. Nevertheless, it is better to receive this kind of information than preventing the user from getting any data.

ACID transactions are avoided throughout the system, because of the High Availability (HA) requirement.

4.1 Partitioning

The fundamental level of consistency is the patient. Data is modelled per patient. In addition to that, client systems often fire off a string of operations on the same patient in a short interval of time, and need read-your-writes consistency. Therefore, the network is partitioned in DCs, such that a given client system is usually operating on the same network partition. Within this partition (DC) the system (almost) provides read-your-writes consistency. This is not the case for the global system.

Given that any citizen can be treated in any part of the country at any time, the medicine card database cannot as such be split into independent parts. On the other hand, no consistency is assumed across person IDs. The set of objects is split into data of different "flavours" (domain data, write-twice request log for caching purposes, write-once access log for auditing purposes).

4.2 Eventual Consistency

Because of the asynchronous client system interfaces, and distributed data centres, two doctors can prescribe conflicting medicine to the same patient simultaneously. A real-life example of this is right after a patient is discharged from hospital and visits his GP. The medicines that a patient was prescribed in the hospital are sometimes carried over from the hospital patient journal to FMK after his discharge, and can coincide with the prescription of new medicine by a GP. Because the system is EC, it is not always visible, that all updates have not yet propagated to all the nodes. This means that conflicts can be detected after the conflicting changes were made. "Conflicting medicine" may be multiple prescriptions of drugs containing the same active substance, or two drugs which interact poorly. Optimally, a doctor making or adjusting a prescription has full overview of the patient's existing prescriptions when he/she does so.

4.3 Offline mode

In normal operation, a hospital can temporarily take ownership of a patients data, while he is admitted. When the hospital has this ownership, it operates offline, until it returns ownership to the central service. In a failure scenario, some client systems must also be able to run offline without connection to a central service.

5 Conflit-free Replicated Data Types

CRDTs have been proposed as an approach to design replicated shared data in distributed systems [6]. The consistency model that a CRDT ensure is "*Strong Eventual Consistency*". It roughly means that, as soon as replicas have delivered the same updates, they have equivalent state. Thus no further message (e.g. rollback, conflict-resolution, ...) are needed. CRDTs has been created to ensure that we have a storage model for handling data which accommodate data's nature in a world of vast communication and no definite central place of storage. Applying an RDBMS for these type of systems can result in systems where a large amount of the processing is required to circumnavigate the shortcomings of a model that is not fit for his purpose. A significant project for researching and mediating this dilemma is SyncFree [2].

CRDT seems an adequate approach when data, as viewed by users, is mutable [7]. Particularly, the FMK system needs a 2-dimensional temporal model, as it allows "old versions" to be updated to reflect that the system does not reflect the real data. The most up to date data is out in the real world. It could be that wrong information was entered into the system (maybe because a secretary entered information from an incomprehensible voice dictation), then the information is updated/decided upon by a third party, after which it needs to be possible to go back and correct the original input.

The structure of a person record is a composite CRDT like this:

- person data (name, id, address, ...)
- treatments (set)
- prescriptions (set)
- prescription events (add-only set)
- other events (add-only set)

Each set is managed by a set CRDT. Different sets CRDT exist, and we use different behaviour/implementation considering the nature of updates. Theses sets may contains mutable or un-mutable information. In case of mutable information, the elements are them-self CRDT.

5.1 Merging

CRDT requires to merge concurrent updates on the data, and thus must deal with potential tensions between them. Merging multiple sets of prescriptions must always converges towards the same set - no matter in which order the operations on the prescription set was done.

A possible conflict occurs in the case when a patient is involved in several treatments (at different facilities or with different health care professionals), but the drugs prescribed for these treatments have undesired interactions. Such "drug incompatibilities" cannot reliably be described as system invariants; they can, to some degree, be flagged by various external expert systems. In the end, the system can only flag non-causal updates, and let the domain professionals decide. One of our working rules when working with drugs is to not make decisions in code, but help the domain professionals to make the decisions.

The core "problem" is what happens when a prescription record is updated with conflict, as it cannot be completely reconstructed from the corresponding "prescription events" (these might have been a better model to be completely ops-based). Prescriptions are not deleted but "seponated" (marked deleted). As mentioned above, in case of write conflicts, we do a "best effort" merge, and flag the record as in need of review. End-user systems are then required to visibly signal this status until a professional review has been performed.

5.2 Invariants

Since availability and partition tolerance are paramount, conflicts cannot be prohibited, but are instead corrected. There is no "strong invariant" in the system meaning the prohibition of data conflicts. The closest thing is that automatically resolved medicine cards must be marked as such (with a "not reviewed" flag) and stay so marked until reviewed and unflagged by a healthcare professional. Prescriptions should not be created using medicines that are not on the list of permitted medicines.

6 Improvements

Although the FMK System is already in production at a nationwide level, some improvements could be achieved.

6.1 Divergence visibility

It would be very useful to have an indication on how divergent the different components are for a given patient. When a doctor sees a patient's data, it is interesting to know if the latest data from, for instance a hospital, is present. This could be represented by a timestamp for which the client knew that all other core-components had successfully completed replication of incoming data. There could always be delayed data from the un-bounded set of peripheral client-systems. Ideally, there would be a timestamp available from all relevant components telling when they last synced data for this patient.

Probabilistic measurements on divergence would be relevant for the technical staff, but provide little direct value to end-users.

6.2 Divergence guarantees

For the usual high availability use cases of providing access to a patient's medication data, it would not be relevant to have mechanism that guarantee that data is not divergent because enforcing consistency requirements reduce availability.

For the offline data analytics use case, it could be relevant not to proceed with the data analysis, until all data is guaranteed to be sufficiently up-to-date. A probabilistic value would be better than nothing, but would increase complexity, because late updates would need to be detected and handled by for instance redoing the analysis.

6.3 Pseudo-Transactions

A common use case involves a pharmacy querying for prescriptions that a patient will be picking up soon (prescriptions stored in the system can be tagged by a pharmacy where it is to be fulfilled). Since this is a very common query, we would like to be able to service such queries with a materialized view, rather than a coverage query which usually involves all machines in a cluster (Riak 2i). This however, is difficult as Riak provides no cross-object consistency. The atomicity/consistency we need can be described as a delayed-consistency materialised view: whenever a pharmacy-prescription relation is added, updated or removed, that will eventually, with a reasonable delay, be reflected in the view. This implicitly addresses the "isolation" part as well: seeing the updated prescription but an unupdated view is acceptable, especially since no high-level operation refers to both.

Adding transactions to Riak would allow updating a materialised view like this, but at the same time reduce the availability, since an update now involves twice as many nodes. With more dependent updates, this can quickly require

that at most a cluster is up to perform the updates. It would be very attractive (and a great research target) to have an alternative model for "causally dependent updates" in which events are reliably - albeit eventually - delivered to other entities inside the storage system, such sends being triggered by writes.

In the case in question, when a prescription is added/updated for a "patient object", imagine a reliable message <add-prescription P> to the "pharmacy object" being en-queued as part of the update. The patient update operation can return success to the caller as soon as the message is reliably en-queued, as it is guaranteed that it will eventually be delivered to the "pharmacy object", even if the node(s) hosting it are currently inaccessible. Especially when the target of such an eventual, but reliably guaranteed, message is a CRDT. This model seems very attractive as an alternative to traditional transactions because it will in many cases be able to gracefully handle out-of-order delivery of operations.

Also, by avoiding transactions spanning multiple entities on the critical path this could lead to significantly better availability than a transaction based model. A guaranteed message delivery system would requires some kind of transactions internal to the storage cluster, but those can be decoupled from the primary read/write operations. Issues can arise if too many messages end up being undelivered for an extended period of time.

A reliable messaging system is much easier to implement if the target of the messages is known to be idempotent, because the system only needs to guarantee "at-least-once delivery." The intersection of CRDT/messaging also brings to mind all the work being done in the area of Command Query Responsibility Segregation (CQRS). Likewise, a good reference for inspiration in this area is Pat Helland's [3].

6.4 Acting on Conflicts

When developing the present implementation of materialised views, we found a need to detect Riak objects with conflicts (i.e. with siblings). Conflicts may appear in ways not triggered by client reads or writes, so objects which are seldom or never read can exist for a long time in a non-resolved state. At present there is no mechanism for querying Riak for conflict-ridden objects and thus to consistently act on conflicts; a possible solution (in the form of a new feature) could be to use Secondary Indexing (2i).

7 Conclusion

The presented FMK system is server centric although it was kept in mind throughout the development that the truth is always what happened in relation to the patient. What is recorded in the databases is an approximation of reality. Capturing the information provided by the users has a priority over ACID. In the case of network partition in any DC response will be managed based only on its own information without trying to fix consistency issues. In such a healthcare system, ACID properties cannot not be ensured. Not because

it cannot be technically achieved but by doing it we would be at risk of loosing or delaying real and vital information.

So we need to infringe the ACID properties and not only for the high availability requirement. Object composition is avoided and transactions does not exist. Divergence and lack of data quality is accepted at the point of data capture. Various mechanism are in place to catch inconsistency on a report level which is then handled by qualified professional staff.

Acknowledgements: This has been created as part of the European research project SyncFree.

References

[1] Danish Helath and Medicines Authority. FMK. http://sundhedsstyrelsen.dk/en/medicines/medicine-profile, 2013.

[2] European research project, grant agreement 609551. Syncfree "large-scale computation without synchronisation". http://syncfree.lip6.fr, October 2013.

[3] P. Helland. Life beyond distributed transactions: an apostate's opinion. In *CIDR*, pages 132–141, 2007.

[4] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't settle for eventual consistency. *Commun. ACM*, 57(5):61–68, May 2014.

[5] D. Pritchett. BASE: An ACID alternative. *Queue*, 6(3):48–55, May 2008.

[6] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976, pages 386–400, Grenoble, France, Oct. 2011.

[7] S. Weiss, P. Urso, and P. Molli. Logoot-undo: Distributed collaborative editing system on p2p networks. *IEEE Transactions on Parallel and Distributed Systems*, 21:1162–1174, 2010.