SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 757
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 77–83 P.

Towards Formalization of a Method for Data Consistency Support in Mobile Ad-hoc Distributed Systems[1]

Maxim Galkin
University of Saint-Petersburg, Universitetsky pr. 28, Peterhof, Saint-Petersburg, Russia
maksim.galkin@gmail.com

Abstract. This paper presents a research-in-progress report on a method for data consistency support in ad-hoc mobile distributed systems. The method is based on a concept of high-level operations "compatibility" and operation history reconciliation with compensations. The described approach is not bound to any particular data storage or communication technology, but in this paper we use tuple spaces as a possible practical medium, which hides network details and mobile nodes heterogeneity. Limits of applicability of the method are described, model and operations are formalized, detailed reconciliation algorithm and communication protocol are discussed.

Keywords: data consistency, ad hoc distributed systems, mobile systems.

1 Introduction

In this paper we present a new method for data consistency support in an ad-hoc mobile distributed system. This method combines some of the optimistic practices known for traditional and nomadic distributed systems (like high-level operations "compatibility" and compensational operations) with special structures such as list-accumulators, which abstract the idea of an operation history [10] (see Section 3). The proposed method specifically deals with reconciliation conflicts separating them into a compensational accumulator, which is only required for external operations, but not necessary for merging. In this paper we also propose using loose communication mechanisms such as tuple spaces, which can hide the intermittent mobile network connection and mobile nodes heterogeneity.

Many researchers contributed to the development and optimization of consistency support methods for traditional client-server and nomadic distributed systems [1, 2, 10, 15]. Most of such methods though rely on some sort of centralized "clusters of consistency" [13], or "master replica", that is a node or a set of nodes, which keeps the latest and most actual data. Nomadic nodes synchronize their state with those

[1] This work was partially supported by Russian Foundation for Basic Research RFBR, grant 10-07-00156.

central nodes and achieve durability for the accepted transactions. Such approach is not applicable in the ad-hoc mobile scenario [3, 9]. Other works that propose weak consistency protocols for distributed systems [6, 8] do not describe any specific conflict resolution protocols, assuming that the conflicts are "unlikely to happen" or giving them only a short remark, though conflict resolution is usually the most complex part of an optimistic system[16]. Instead of avoiding conflicts through locks or quorum protocols (impossible in a disconnected scenario) or ignoring them we have to embrace consistency conflicts as a part of the usual working scenario and give away the durability property. We propose a specific mechanism for handling consistency conflicts with compensational accumulator, which is discussed in details in Section 3. This mechanism supports automatic semantic conflict resolution as well as delayed manual.

On the other hand, one of the strong advantages of ad-hoc distributed systems is in the absence of a "single point of failure". All of the network nodes are equally important, and if we can keep their replicas consistent then our system can continue working even if it loses part of the nodes. Another advantage is the potentially unlimited extensibility (i.e. the ability to add and accommodate new network nodes) as there is no bottleneck like main server performance or main server connection bandwidth. Every node is only going to "talk" to its neighbors. Finally, in a mobile world it's a possible scenario that the known main servers can be inaccessible, while the neighbour devices can be still available. Ideally such availability can be exploited to support the same level of services the node had while it was connected to the static infrastructure. In practice our method can only guarantee limited level of services, as they must comply with the conditions we describe in Section 2 and should not have many conflicting operations. The exact practical limitations are to be verified in the experimental part of my Ph.D. thesis, which is yet in progress.

The proposed method is indifferent to an underlying data model, only operations are important. It demands all nodes to be aware of:

- some common initial data state S_0, which can be just an empty set, or a "map" over which the nodes work;
- a set of high-level operations O=M+C, where M are model operations recorded by nodes and C are compensational operations;
- a compensational function δ: <M> $\rightarrow$ <C>, which maps model's list-accumulator to a compensational list-accumulator;
- a reconciliation operation R.

This is discussed in detail in Section 3. Having that defined the nodes become autonomous and start applying operations from M to their local replicas. At some random (unpredictable) points in time nodes perform binary reconciliation operation R of their model accumulators and through this achieve a consistent and actual data.

$$<M> \text{ `R' } <M> \rightarrow <M> . \quad (1)$$

Due to the general approach this method can be used to build a middleware platform for consistency support in ad-hoc distributed systems.

2 Method Applicability

In this section we discuss two conditions of the proposed method applicability.

First of all, let's clarify what we mean by "mobility" of the network nodes. Basing on the classification given in [11] we assume our nodes: have intermittent network connection; have dynamic location / dynamic context; need to discover other hosts in an ad-hoc manner; are heterogeneous. We abstract from nodes heterogeneity and context change by using loose generic tuple spaces communication mechanism. Also we do not consider our nodes to have any significant limitations in CPU / power resources for the sake of this research, due to the rapid development of mobile devices and because we not only take PDAs and communicators into account, but laptops as well.

The second applicability condition concerns the reconciliation operation 'R' properties. This operation is naturally commutative, because of the system symmetry: all nodes histories are equally important and there is no point in making the reconciliation to prefer one argument over another. What we have to additionally demand is the associativity property for R. Without that property the result of the reconciliation will depend on the particular order of nodes reconciliations thus, as we cannot guarantee any particular order, making the result non-deterministic. For some certain sets of operations and compensations it might be possible to bound this non-determinism, so that, for example, any possible order of R's would return a "consistent enough" result, but this topic is out of the scope of this research. So, we demand R to be commutative and associative.

3 Data Model and Operations

For the sake of the proposed method, a set of defined high-level operations over the data is more significant than the data model itself and further in this paper we will only discuss operation set properties. At the same time we must note that in some related works [10, 14] special data model was crucial in defining such operations, which cause less conflicts and therefore better suitable for collaborative work.

In general, we can name two sides or two states of the data in the system. One side is the real data, which resides in the mobile agents, and which is continuously updated by them and another side is the ideal data, that can be achieved by a full semantically-correct reconciliation of all data replicas. In the process of work our cloud of local data replicas strives to become closer to the ideal state, where every agent knows the current and actual state of data. Of course, in reality the degree of consistency depends on many factors, like the connection quality between agents and the intensity of the continuous local updates. We plan to examine the ratio of these factors in the experimental part of the work.

To formalize that we will use a variation of the version vector approach [16]. Assuming we have N nodes and each node has a version number, which is incremented on every update this node commits, the state of a node i can be expressed as a vector Vi, where Vi[j] is the last version of node j known to node i. Then an ideal fully consistent state would have every Vi[j] = Vj[j], if by Vj[j] we mean the global latest version of node j. Every state different from that would have some node divergence and require

further reconciliation. We must note here that the actual node data is fully determined by its version vector due to the operation 'R' properties, discussed in Section 2.

In most cases, a table of compatibility conditions is filled together with operations definition. In the CoACT model [15] such conditions are called "activity interleaving rules" and given in a form of predicates for each pair of operations. In general, operations compatibility may depend on both their nature (e.g., two "read" operations will never create a conflict) and data they are applied to.

To successfully maintain consistency the set of operations for ad-hoc mobile distributed system must ensure as few conflicts as possible, in other words operations must be as compatible as possible, because every conflict in such a system is a serious problem. Two mobile nodes, which discover a data inconsistency during their interaction don't possess any advantage over each other, unlike it happens in "nomadic" distributed systems, where the fixed nodes have the priority, the most consistent state. Usually in ad-hoc distributed system it's also impossible to draw a human operator into the process of conflict resolution, because the inconsistent state can be discovered at nodes that aren't the authors of the conflicting operations or when the authors are offline [16].

Therefore every conflict resolution must be automatic in accordance with some pre-defined rules of reconciliation, as the system can't be blocked by a conflict. Such rules can be of two major types: differential (e.g. when the system takes one of two conflicting operations basing on its timestamp) and integral (if the system, for example, has some pre-defined objective function or cost function over the data model and it discards one of the conflicting operations basing on the value of the function). The system should also allow delayed manual conflict resolution as this is the only ultimate way to resolve possible operation conflicts.

In this paper we propose using two list-accumulators for each data object to meet the described requirements. Accumulators are intended for "intervention" of the application to the conflict resolution. As against simple types, the accumulator stores operations applied to some initial value of simple type instead of the explicit value of the element. Thus, accumulators allow on the air replaying the operations [10].

The first accumulator <M> is used to store the model operation history over a data element, and additional accumulator <C> is used to store the compensation operations over data elements. On reconciliation 'R' we merge only model accumulators to propagate the known updates throughout the system. After that, nodes use the compensational function δ to calculate the compensations <C> for the new model history <M>. Compensations may include simple cancellations of some operations from the main history, or those can be additional commands to semantically resolve a conflict in <M>. <C> is constructed in such a way, so it can be easily merged with <M> for external queries. But another important usage of <C> is that it can be examined separately by client applications and used to show the collected conflicts to a user for further reconciliation (the user, for example, might want to undo or redo one of the cancelled operations with the newer version of the object).

Formally, the reconciliation operation of nodes i and j can be described as maximization of each version vector component separately, so that

$$Vresult[k] = \max(Vi[k], Vj[k]), \text{ where } k = 1..\ N\ . \qquad (2)$$

4 Transactional Properties

The above-stated implies that even if some operation has been committed at one of the nodes and lived in the system for some period of time it is not guaranteed to be fixed in the system forever as one of the other nodes could have submitted a conflicting operation that will eventually overcome the first one. In other words, the Durability property from the ACID set is not guaranteed by our system for the sake of Consistency. Here we can also notice that in an ad-hoc distributed system no "global commit" or "strong" [13] operations are possible. On the contrary, all operations in the system are "weak", that is they are only applicable to the local replica of the data and they are only guaranteed to be "safe" until next data reconciliation with another node.

The Consistency property in our system should not to be confused with "single-node" data consistency. This type of consistency is always guaranteed in ACID systems, no transaction is allowed to leave the data in an inconsistent or partially consistent state as it would make the data meaningless. Our high-level operations can be seen analogous to such classical transactions as they are supposed to always leave our model semantically correct. So when we talk about the "degree of consistency" and node divergence we mean distributed system consistency. As our consistency support techniques cannot be based on locking in the mobile environment, they can only be optimistic, hence we have weak consistency type of system [6]. We can guarantee eventual type of weak consistency only under certain conditions, as we cannot always expect full convergence of the nodes' states, unless they participate actively in the communication. Another problem for the convergence is the possibility of distributed network partitioning, where some nodes from one network part will never be exposed to nodes from another. This limits the applicability of the proposed method in cases where eventual consistency is demanded.

With respect to the other ACID properties such as Atomicity and Isolation we should notice two levels of their applicability. Again if we consider our high-level operations as a specific form of transactions consisting of some elementary read/write sub-operations, then these transactions will satisfy both A and I properties. If on the other hand we introduce even higher-level transactions that consist of our original operations we will need to give away the highlighted properties, similar to how it happens in the "sagas" transaction model [5]. Otherwise, long-running isolated transactions will inevitably increase the number of conflicts just like it is observed in other types of distributed database systems [7]. In our system this can lead to increased load on mobile nodes and the overall decline in data quality, as compensative transactions can sometimes be ineffective in cleaning up the database due to lack of transaction isolation.

5 Nodes Interaction Protocol

We assume that numerous mobile nodes interact with each other by establishing connection with some of the other nodes in their "visibility range" and sharing a tuple space, for example, LIME [12] or JavaSpaces [4]. The tuple space has an ID, which is bound to the data model, so every node can actually "host" several models and reconcile them in different tuple spaces.

At one time there can be several nodes sharing one tuple space, but only one "editor". The editor is selected in some deterministic and reliable way, for example, by passing a token from one node to another. Tuple space contains the model achieved so far after the reconciliation of all the previous "editors" models. Current editor's task is to reconcile it with his own model, if necessary, update its own model with the shared model and pass the token to the next node. Every node can go offline any time, if the "editor" disconnects a new token must be generated.

As the implementation of the model reconciliation operation 'R' we propose simple timestamp-based ordering of the operations. It's easy to observe such 'R' is both commutative and associative.

After the reconciliation we need to calculate the compensational accumulator <C>, this can be done on each node separately or once per tuple space if one node is selected as compensation accumulator "editor". One obvious further enhancement of the protocol is, of course, in keeping the old compensations unchanged if there were no model changes in the compensated area.

In this scenario the common tuple space can be found to correspond to a notion of a "cluster of consistency" [13], with the only difference that in our case no data in a cluster is durable. For example, if two nodes from different clusters interact any of the operations in their replicas can be potentially conflicting and subject for removal.

6 Conclusion and Further Work

We have introduced a method for consistency support in a mobile ad-hoc distributed system, based on the reconciliation process of high-level operations histories. We also presented a description of supported data models and the requirements for operations set. We have analyzed the transactional properties of the proposed system and outlined a protocol for nodes interaction.

We plan to further develop this method with regards to an experimental proof of concept. In future theoretical work we plan to introduce metrics of the global consistency in such a mobile system based on the version vectors to be able to evaluate the efficiency of the method and/or compare our method with other methods. Another way of applying such metrics is to use them in the described mobile system as an integral rule for operational histories reconciliation, thus we can sort out nodes, which have their replica so obsolete or inconsistent that it doesn't make sense to compare them with the current data.

One more possible further research direction is to look at weaker reconciliation operations 'R', which do not require the associativity property.

7 References

1. Adya, A.: Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions. PhD thesis at the MIT, certified by Barbara Liskov (1999).
2. Bosneag, A-M., Brockmeyer, M.: A Formal Model for Eventual Consistency Semantics. In: Proc. of the Parallel and Distributed Computing and Systems Symposium (2002).
3. Capra, L., Emmerich, W., Mascolo, C.: Middleware for Mobile Computing: Awareness vs. Transparency. In: Int. 8th Workshop on Hot Topics in Operating Systems (2001).

4. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces, Principles, Patterns and Practice. Addison-Wesley (1999).
5. Garcia-Molina, H., Salem, K.: Sagas. ACM SIGMOD Record, Volume 16, Issue 3, pp. 249-259 (1987).
6. Golding, R.: A Weak-Consistency Architecture for Distributed Information Services. Computing Systems Journal, vol. 5 (1992).
7. Gray, J.N., Helland, P., O'Neil, P., Shasha, D.: The Dangers of Replication and a Solution. In Conference on Management of Data, pp. 173-182, Canada (1996).
8. Gustavsson S., Andler, S.F.: Self-stabilization and eventual consistency in replicated real-time databases. In: Proc. of the first Workshop in Self-Healing Systems. WOSS '02, Charleston, SC, USA (2002).
9. Imielinski, T., Badrinath, B.R.: Mobile wireless computing: Challenges in data management. Communications of the ACM, Vol. 37, No. 10, pp. 19—28 (1994).
10. Kozlova, A., Kochnev, D., Novikov B.: The Middleware Support for Consistency in Distributed Mobile Applications. In: Proc. of the Baltic DB&IS'2004, 145-160, Riga, Latvia, Scientific Papers University of Latvia (2004).
11. Mascolo, C., Capra, L., Emmerich, W.: Principles of Mobile Computing Middleware. In: Mahmoud, Q. (ed): Middleware for Communications, John Wiley (2004).
12. Murphy, A. L., Picco, G. P., Roman, G-C.: LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. ACM Transactions on Software Engineering, Vol. X, No. X, pp. 1-48 (2006).
13. Pitoura, E., Bhargava, B., Wolfson, O.: Data Consistency in Intermittently Connected Distributed Systems. In: Transactions on Knowledge and Data Engineering (1999).
14. Proskurnin, O., Novikov, B.: Towards collaborative video authoring. Proc. of ADBIS'03, 370-384, Dresden, Germany (2003).
15. Rusinkiewicz, M., Klas, W., Tesch T., Wasch, J., Muth, P.: Towards a Cooperative Transaction Model: The Cooperative Activity Model. In: Proc. of the 21st VLDB Conference, Zurich, Switzerland (1995).
16. Saito Y., Shapiro M.: Replication: Optimistic Approaches. Internet Systems and Storage Laboratory, HP Labs, Palo Alto (2002).
17. Weikum, G., Schek, H.-J. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In: Elmagarmid, A.K. (ed): Database Transaction Models for Advanced Applications, Morgan Kaufmann (1992).