Data Partitioning through Integrity Constraints

Lorenz Froihofer, Markus Baumgartner, Johannes Osrael, and Karl M. Goeschka
Vienna University of Technology
Institute of Information Systems
Argentinierstrasse 8/184-1
1040 Vienna, Austria
{*lorenz.froihofer|johannes.osrael|karl.goeschka*}*@tuwien.ac.at*

Abstract

Distributed systems that are subject to network partitions often face the conflicting requirements of high availability and strong consistency. While optimistic replication protocols address the problem based on the assumption that conflicting (write) operations are rare, data partitioning allows to prevent conflicts at all through (non-redundant) distribution of data items throughout a system. Within this paper we contribute with a middleware-supported approach based on explicit runtime data integrity constraints that allows for a combination of optimistic replication protocols and data partitioning. Based on an example, we show how this concept can be applied in order to gain high availability while still limiting the amount of inconsistency introduced to a system.

1. Introduction and related work

Network partitions have been subject to extensive research for many years and interesting results have already been achieved. Already in 1985, Davidson et al. [3] contributed with a survey about *consistency in partitioned networks*, where they discussed several optimistic and pessimistic strategies to address the conflicting requirements of availability on the one hand and correctness (integrity, consistency) on the other hand. Actually, this interdependence is stated more precisely by the CAP principle [4, 7], specifying that a system can only fully satisfy two of the three properties: **C**onsistency, **A**vailability, and **P**artition-tolerance (strong CAP principle). However, the weak CAP principle already defines that *the stronger guarantees are provided for two of these properties, the weaker guarantees can be provided for the third*. Obviously, there is a trade-off between these requirements. This trade-off potential is addressed, for example, through data partitioning [2] and optimistic replication [8] techniques.

Data partitioning techniques address network partitions through (non-redundant) distribution of data items across the system. For example, tickets of a ticket booking system might be distributed across nodes, which subsequently can sell their amount of tickets individually. Of course, redistribution of tickets between the nodes is possible in order to avoid one node running out of tickets. However, according to the CAP principle, this behaviour relaxes the availability requirement as during a network partition, some partitions might run out of tickets while other ones might still have several of them. Optimistic replication techniques on the other hand relax the consistency requirements based on the assumption that (write) conflicts are rare and therefore the benefit gained in higher availability is worth the effort caused through conflict resolution.

However, viewing data items on their own is often not sufficient, as data are generally subject to several correctness criteria, including data integrity constraints that represent application requirements. Consequently, we proposed and implemented a middleware-based approach for adaptive dependability [1] by balancing availability and integrity through the use of explicit runtime constraints [5]. Within this paper, we contribute with an approach to use explicit runtime constraints in order to address network partitions through a combination of data partitioning and optimistic replication strategies.

2 Partition-sensitive constraints

In order to support an application with the balancing of availability and integrity, we require applications to provide the data integrity constraints within explicit constraint checking classes, one class per constraint. A constraint implementation has to adhere to a predefined interface and implement a validate(...) method, providing true if the

constraint is satisfied and false, if it is violated. While the validation of such a constraint is straightforward to implement, the maintenance of consistency with respect to the constraints is a challenging task within partitionable systems.

Imagine a distributed replicated flight booking system, for example, with an integrity constraint (*ticket-constraint*) stating that the system must not sell more tickets for a flight than available seats in the airplane. Within this system, we have an airplane with 80 seats of which 70 are already booked. The system now suffers from a network partition. Due to the high availability requirement for this system, we allow write access in different partitions, temporarily accepting potential inconsistencies. Assume that customers buy seven tickets in one partition, which now has a total of 77 sold tickets. The ticket-constraint is satisfied in this partition. Subsequently, customers buy eight tickets in the other partition, leading to 78 sold tickets. The ticket-constraint is also satisfied in this partition. After the network partitions are reunified, the system has to reconcile the updates made in the different partitions, effectively leading to 85 sold tickets in total. Consequently, our ticket-constraint is now violated. To solve this issue, five customers will be rebooked to another flight (compensating action).

The previous example shows that although a constraint is satisfied in degraded mode while node or link failures are present, it might be violated afterwards when the system recovers from a previous failure. Consequently, constraint validation is not reliable during degraded mode and we are only able to determine that a constraint is possibly satisfied, possibly violated or even uncheckable if the constraint requires unreachable objects for validation. We call such situations a *consistency threat* [5].

While rebooking five passengers to another flight solves the inconsistency, it would be desirable to not introduce such an inconsistency at all. For some applications, where the data can be partitioned like the tickets in the flight booking example, a significant improvement is possible. Our middleware makes use of group membership (GMS) and group communication (GC) primitives in order to implement the replication support. Similar to Gifford's solution of weighted replica copies [6], we allow the association of weights with server nodes. The GMS component can thereafter calculate the weight of the current partition in relation to the whole system. This value is provided by the middleware to the application in order to take the current partition weight into account for constraint validation, effectively leading to *partition-sensitive integrity constraints*.

Based on these prerequisites, data can be partitioned during runtime by using partition-sensitive constraints. The ticket-constraint, for example, saves the number of tickets sold in healthy mode. In degraded mode, it partitions the number of still available tickets t (number of seats minus number of tickets sold in healthy mode) according to the partition weight, effectively leading to t_x available tickets for a partition x ($t = \sum_{x=1}^{n} t_x$). The constraint will only be satisfied within partition x, if the number of tickets sold during degradation is below or equal t_x. This improves behaviour in degraded mode and introduces almost no inconsistencies based on the fact that tickets are mainly sold and rarely cancelled. In the best case, no inconsistencies are introduced at all, although write access in different partitions is possible.

3 Conclusion

Partition-sensitive constraints support a combination of optimistic replication and data partitioning in order to reduce the amount of inconsistency introduced into a distributed system while network partitions are present. This reduces the amount of inconsistencies to be cleaned up while the system reconciles the updates performed in different network partitions during degraded mode. Consequently, partition-sensitive constraints can increase the performance of the reconciliation phase.

Acknowledgements. This work has been partially funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract 004152, `http://www.dedisys.org/`).

References

[1] A. Avižienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[2] U. Çetintemel, B. Özden, A. Silberschatz, and M. J. Franklin. Design and evaluation of redistribution strategies for wide-area commodity distribution. In *ICDCS*, pages 154–161, 2001.

[3] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in a partitioned network: a survey. *ACM Comput. Surv.*, 17(3):341–370, 1985.

[4] A. Fox and E. A. Brewer. Harvest, yield and scalable tolerant systems. In *Workshop on Hot Topics in Operating Systems*, pages 174–178, 1999.

[5] L. Froihofer, J. Osrael, and K. M. Goeschka. Decoupling constraint validation from business activities to improve dependability in distributed object systems. In *Proc. 2nd Int. Conf. on Availability, Reliability and Security*. IEEE CS, 2007.

[6] D. K. Gifford. Weighted voting for replicated data. In *SOSP '79: Proceedings of the seventh ACM symposium on Operating systems principles*, pages 150–162, New York, NY, USA, 1979. ACM Press.

[7] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[8] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.