Brief Announcement: Fast Travellers: Infrastructure-Independent Deadlock Resolution in Resource-restricted Distributed Systems

Sebastian Ertel[1], Christof Fetzer[1], and Michael J. Beckerle[2]

[1] Technische Universität Dresden
Dresden, Germany
firstname.lastname@tu-dresden.de
[2] Waltham, MA, USA
michael.beckerle@alum.mit.edu

Introduction. In the area of data integration and middleware, distributed data processing systems create directed workflows to perform data cleansing, consolidation and calculations before emitting results to targets such as data warehouses. To provide fault tolerance, expensive system-wide checkpoints of distributed workflows want to be performed on the level of seconds while commits to transactional target resources must happen much more frequently to satisfy near real-time result latency [1] and small transaction size requirements. When there exists non-determinism in the workflow, the commit against a transactional target is allowed to be issued only when the determinants were saved to stable storage and deterministic replay can assure exactly-once result delivery. That is, there exists a *dependency*: the process q (a.k.a. operator or component in the context of data integration) executing the transaction is not allowed to make forward progress unless it has received the notification of the non-deterministic process p stating that the results to be committed can be replayed deterministically in the event of a crash.

The Deadlock Problem. Two challenges exist: 1) the limited system view of the processes and 2) their resource limitations. The first challenge requires a process to have no knowledge about the workflow it is contained in; a common distributed system model aspect [2]. Therewith, creating new connections especially for the above dependency is neither favourable nor possible. A solution based on already existing FIFO channels defined and maintained by the system is desirable. Respectively, the distributed algorithm to coordinate the commits, sends the notification, a *marker* m, in-order with the data. But in between p and q, the data stream can be enriched with a theoretically unbounded number of new messages. In contrast to that, the second challenge refers to the fact that a transaction at q is restricted to a maximum size, modelled by input buffer I_q, while process p only has a limited output buffer Q_p to fulfil latency requirements. Hence, it can not be assured that m arrives at q in the interval $|I_q|$.

Fast Travellers. We solve the above Deadlock Problem by extending our system model such that a channel supports out-of-band message transmission,

as known from TCP out-of-band. Respectively, we classify markers with respect to their channel transmission characteristics.

- *Slow Travellers (ST)* are markers in the classical sense that travel through the channels in-order with all other messages (as described in the distributed snapshot algorithm [3]).
- *Fast Travellers (FT)* are markers that are transmitted out-of-band with respect to all messages among a channel.

To always enable the receipt of a Fast Traveller, we state that every process leaves one spot available in its input buffer at any time. The solution for our deadlock problem obviously suggests that the marker m has to be a Fast Traveller.

Assuring Correctness. But as a matter of fact, it is essential to the correctness of most marker-based algorithms that the marker actually travels in-order with the data/messages. For example, the deterministic replay algorithm requires that no messages that can not be replayed deterministically are committed to the transactional resource. This reasoning is based on the "happened before" relationship of message arrivals in the distributed snapshot algorithm [3]. There, the receipt of a Slow Traveller at any two processes p and q with state s_p and s_q marks these states as *computationally equivalent*; $s_p \equiv s_q$. We also define the state s_q of process q when the marker was not received yet as *computationally before* ($s_q < s_p$) A distributed algorithm is correct iff the delivery of a message, sent by p after the m was sent in state s_p, is disabled at state s_q, where $s_q \leqq s_p$.

Marker Pairs. Therefore, we combine the two traveller types such that the creation of a marker at process p produces two messages: 1) a Fast Traveller ft to resolve deadlocks and optimize the resource usage among a target process q and 2) a Slow Traveller st to preserve the correctness of the algorithms. Whenever process q receives ft and adds it to I_q, it holds that q's current state $s_q < s_p$ due to transmission of the messages among channel c. Furthermore, it holds that q's subsequent states up until the arrival of st are computationally in the past of s_p and therewith actions in q depending on ft are enabled. The receipt of st, where $s_p \equiv s_q$, only evicts ft from I_q in order to disable actions depending on ft again and assure correctness of the marker algorithm.

We used our Marker Pair Algorithm to efficiently solve the above deadlock problem in our data integration system[3] and are convinced that there exist many more use cases for Fast Travellers in a variety of different distributed algorithms.

References

1. Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.: Supporting streaming updates in an active data warehouse. In: ICDE. (2007)
2. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)
3. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. **3** (February 1985) 63–75

[3] http://ohua.sourceforge.net