EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems

Jocelyn Thode
Université de Fribourg
Fribourg, Switzerland
Email:jocelyn.thode@unifr.ch

Ehsan Farhadi
Université de Neuchâtel
Neuchâtel, Switzerland
Email:ehsan.farhadi@unine.ch

***Abstract*—Total ordering has always been of interest in large-scale distributed systems. EpTO is one of the recently introduced total order algorithms for large-scale distributed systems and claims to provide total order and scalability at the same time. In this paper, we verify this claim by implementing EpTO and evaluate its reliability in real-world conditions. We then compare it to a deterministic total order algorithm named JGroups.**

I. Introduction

Creating an algorithm which provides scalability, integrity and validity, along with a total ordering for the events through all peers in a distributed system has been one the hot topics in distributed systems research for many years. One of the recently designed algorithms on this topic is EpTO [1]. EpTO is an algorithm which claims to provide integrity, validity and total order in a large-scale distributed system. In addition, EpTO is designed to work without a global clock, that is why EpTO is well-suited for dynamic large-scale distributed systems.

There are many other algorithms for disseminating and ordering events in a distributed system. There are some deterministic algorithms, which guarantee total order, agreement or other strong properties. Unfortunately, these types of algorithm are not scalable enough to be used in a large-scale distributed system [2, 3].

The problem with existing deterministic total ordering protocols, is that they need some sort of agreement between all peers in the system. This causes a massive amount of network traffic and overhead on the system. Moreover, an agreement feature for an asynchronous system requires to explicitly maintain a group and have access to a failure detector [4, 5]. Due to faults and churn in large-scale distributed systems, failure detector turns into a bottleneck for the system and thus, limits the scalability of the algorithm.

As an alternative to deterministic algorithms, there are probabilistic algorithms, focusing on scalability and resiliency against failures using a probabilistic dissemination approach [6, 7, 8, 9, 10, 11, 12, 13]. These algorithms guarantee the dissemination of events in the system with high probability. This way, there is no need for failure detectors and redundant traffic, making these algorithms highly scalable. Although as these algorithms focus on reliability of dissemination, they often have to ignore other properties such as total ordering.

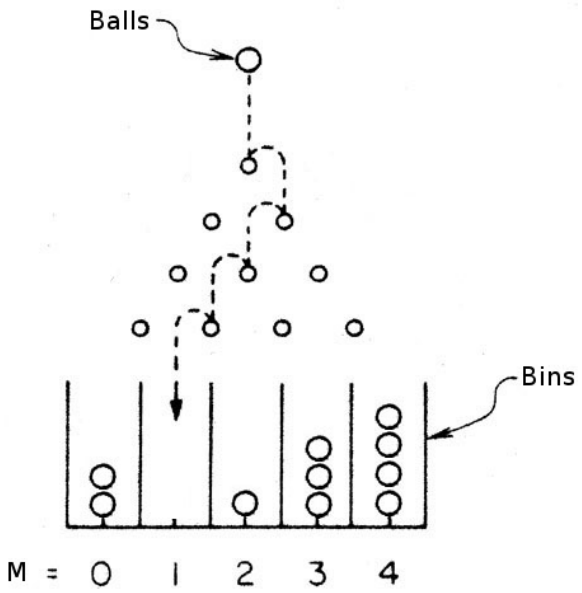

Figure 1. Balls-and-Bins

This is where EpTO comes into light. EpTO, by mixing these two types of algorithms, provides total order along with scalability, validity and integrity. EpTO consists of two distinct parts. The first part is probabilistic dissemination. EpTO guarantees that all peers will receive an event with arbitrarily high probability. The second part of EpTO is deterministic ordering. Once peers have received all events, they will deterministically order them using the events timestamp, and in case of a tie use the event's broadcaster id.

To model the first part, EpTO is using a *balls-and-bins* approach [13]. The balls-and-bins problem is a basic probabilistic problem: consider n balls and m bins where we consequently throw balls into a bin completely random and independent from other balls. In this scenario, one of the natural questions that comes to mind is: what is the minimum number of balls that should be thrown, so that every bin has at least one ball?

Using the balls-and-bins approach we model processes as bins and events as balls. Based on this model we will calculate how many balls need to be *thrown* such that every bin contains at least one ball with arbitrarily high probability. Using this

approach the number of messages transmitted per process per round is logarithmic in the number of processes, therefore the number of messages sent in the system is low and uniform over all processes. Using these approaches, EpTO becomes highly scalable and resilient, while still providing total order.

A. Contribution

Until now, the creators of EpTO have only tested this algorithm in a simulated environment. We are going to implement EpTO in pure Java using the NeEM library [14] and show that EpTO is actually suitable for real-world large-scale distributed systems. Afterwards we will evaluate EpTO by comparing it to the deterministic total order algorithm provided by JGroups [15] in both stable (no-churn) and unstable systems. These comparisons help us verify if EpTO is actually performing as expected in a real-world scenario.

These tasks will be split in two parts. The first part will assume that all peers are known by all peers. We will implement EpTO using this assumption and test it against JGroups. In the second part of the tasks, we will assume unknown membership on all peers. We will update EpTO and use a PSS[1] to find peers to exchange with. We will then update the underlying layer NeEM to use UDP instead of TCP and switch to NIO.2[2].

II. Ordering Algorithms

Distributed systems, like centralized systems need to preserve the temporal order of events produced by concurrent processes in the system. When there are separated processes that can only communicate through messages, you can't easily order these messages. Therefore we need ordering algorithms to overcome this problem.

We have two types of ordering algorithms [3]: the partial order algorithms and the total order algorithms.

A. Partial Order Algorithms

Assuming S is partially ordered under $\leq$, then the following statements hold for all a, b and c in S:

- Reflexivity: $a \leq a$ for all $a \in S$.
- Antisymmetry: $a \leq b$ and $b \leq a$ implies $a = b$.
- Transitivity: $a \leq b$ and $b \leq c$ implies $a \leq c$.

B. Total Order Algorithms

A totally ordered set of events is a partially ordered set which satisfies one additional property:

- Totality (trichotomy law): For any $a, b \in S$, either $a \leq b$ or $b \leq a$.

In other words, total order is an ordering that defines the exact order of every event in the system. On the other hand, partial ordering only defines the order between certain key events that depend on each other. Partial order can be useful since it is less costly to implement. However, in some cases the order of all events is important, for example when we need to know exactly which operations have been invoked in which order, we have to use total order, otherwise we could end up in an inconsistent state.

[1]Peer Sampling Service

[2]Non-blocking I/O available from Java 7

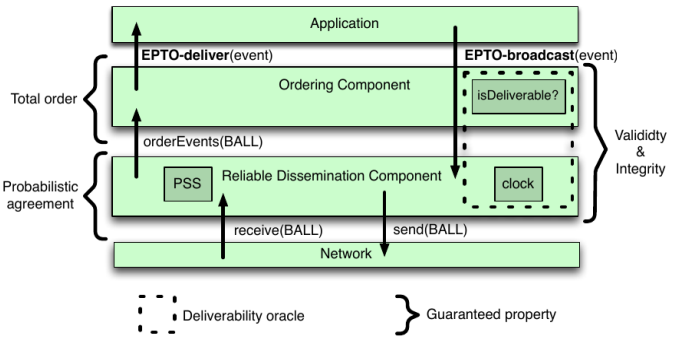

Figure 2.

We need a better quality figure (SVG)

[4].

III. EpTO

A. Definitions

A process or peer is defined as an actor in our system running the application that needs total order. Each process will communicate with other processes in the distributed systems and exchange events and order them together.

An event is defined as something that happens at a given time. For example, we could imagine a system where each process publish some sort of data. The moment where this data is published combined with the data is called an event.

A ball is a set of events bundled together and sent as one package. We use balls in EpTO to reduce network traffic and make it scalable in terms of processes and events.

We define scalable as it was defined in [1]: "The number of messages transmitted per process per round is logarithmic in the number of processes, ..."

Since EpTO is using a probabilistic agreement instead of a deterministic agreement, there might be a situation where a peer does not receive an event (with a very low arbitrary probability). In this case there will be a hole in the sequence of delivered events but even in this case, the order of the delivered event will be protected by EpTO's deterministic ordering algorithm and total order property is preserved.

B. Dissemination Component

The Dissemination component is the component that bridges EpTO with the rest of the network. As we can see in Figure 2, it is in charge of receiving balls, open them , pass them to the Ordering component and then forward them to *K* other processes, where K denotes the gossip fan-out.

When an application wants to publish an event, it will broadcast this event to the Dissemination component.

C. Ordering Component

The Ordering component is responsible for ordering events before delivering them to the application. To achieve this, the Ordering component has a *received* hash table of (*id*, *event*) pairs containing all the events which are received by the peer, but not yet delivered to the application and a *delivered* hash table containing all the events which are delivered.

In brief, the Ordering component first increments the timestamp of all the events which have been received in previous rounds to indicate the start of a new round. Then, it processes new events in the received ball by discarding events that have been received already (delivered events or events with timestamp smaller than the last delivered event). This is done to prevent delivering duplicate events. The remaining events in the received ball will be added to the *received* hash table, and wait to be delivered based on the Stability oracle.

D. Stability Oracle

The Stability oracle is the component that outputs timestamps. It will increment its local clock every round as well as synchronize itself using timestamps of newly received events, to make sure the local clock does not drift too much.

This Stability oracle offers an API to the Ordering component letting it know when an event is mature enough to be delivered to the application.

As this component is local and only correct itself when we receive new events, it generates no network traffic. This means it does not impact the scalability of EpTO.

IV. Performance Comparisons

Here we will present the different test we ran, the methodology used and the result we obtained

A. Methodology

Here we explain how we ran our tests and on what type of machines

B. Peer membership known

We will present the results obtained and try to explain the results

C. Peer membership unknown

We will present the results obtained and try to explain the results

V. Conclusion

He we will summarize the results we found and present some future tasks that could be accomplished on EpTO

Acknowledgment

Write acknowledgment

References

[1] M. Matos *et al.*, "Epto: An epidemic total order algorithm for large-scale distributed systems", in *Proceedings of the 16th Annual Middleware Conference*, ACM, 2015, pp. 100–111.

[2] X. Défago, A. Schiper and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey", *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 372–421, 2004.

[3] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[4] T. D. Chandra, V. Hadzilacos and S. Toueg, "The weakest failure detector for solving consensus", *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 685–722, 1996.

[5] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[6] K. P. Birman *et al.*, "Bimodal multicast", *ACM Transactions on Computer Systems (TOCS)*, vol. 17, no. 2, pp. 41–88, 1999.

[7] N. Carvalho *et al.*, "Emergent structure in unstructured epidemic multicast", in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, IEEE, 2007, pp. 481–490.

[8] A. Demers *et al.*, "Epidemic algorithms for replicated database maintenance", in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, ACM, 1987, pp. 1–12.

[9] P. T. Eugster *et al.*, "Lightweight probabilistic broadcast", *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 4, pp. 341–374, 2003.

[10] P. Felber and F. Pedone, "Probabilistic atomic broadcast", in *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*, IEEE, 2002, pp. 170–179.

[11] M. Hayden and K. Birman, "Probabilistic broadcast", Cornell University, Tech. Rep., 1996.

[12] C. Kim, J. Ahn and C. Hwang, "Gossip based causal order broadcast algorithm", in *Computational Science and Its Applications–ICCSA 2004*, Springer, 2004, pp. 233–242.

[13] B. Koldehofe, "Simple gossiping with balls and bins", in *Studia Informatica Universalis*, 2002, pp. 109–118.

[14] (). Neem : Network-friendly epidemic multicast, [Online]. Available: http : / / neem . sourceforge . net/ (visited on 24/04/2016).

[15] (). Jgroups - a toolkit for reliable messaging, [Online]. Available: http://jgroups.org/ (visited on 24/04/2016).