Exploiting Interest Clustering for Efficient Event Timestamping in Distributed Publish/Subscribe Systems

Roberto Baldoni, Silvia Bonomi, Marco Platania, Leonardo Querzoni
Dipartimento di Ingegneria Informatica Automatica e Gestionale "A. Ruberti"
Sapienza University of Rome
{baldoni|bonomi|platania|querzoni}@dis.uniroma1.it

In the last few years, cloud computing emerged as the mainstream technology to provide on-demand resources as a service over the Internet. Users can access these resources anytime and anywhere, both from desktops or mobile platforms. Amazon Web Services, Google Apps, Microsoft's Azure are just a few examples of cloud infrastructures that provide services ranging from storage and application development to high speed computing platforms.

A public cloud is typically a complex infrastructure composed by one or more data centers, where a huge number of services runs on a large amount of hardware. A typical example is represented by eBay, whose internal architecture has been divided into multiple disjoint subsystems: Users, Items, Transactions, Products, Account, Feedback (vertical division). Each subsystem is further divided into replicated chunks (horizontal division) to parallelize the handling of requests within these [8]. This segmentation was driven by manageability, cost reduction and, most of all, scalability purposes. Scalability, in fact, is considered an overarching goal by cloud providers, and a key factor to achieve it is *decoupling*: cloud nodes should quietly go about their work, avoiding or, at least, reducing interactions, coordination and synchronization [5].

Achieving scalability in cloud computing may come at the cost of data consistency: with reference to the previously described eBay infrastructure, each time an update is performed on a chunk data, the same operation must be also executed on all its replicas that are possibly spread over multiple machines in diverse datacenters in order to improve fault tolerance and data availability. To prevent inconsistencies, a typical approach is to update replicas by means of a locking-based protocol. However, this introduces an unsustainable load due to interactions and synchronization among cloud nodes that may hamper the scalability of the system. This is why major cloud providers are moving towards a decentralized convergence behavior in which replicas are maintained in transiently divergent states, from which they will converge to a consistent state over time. This behavior is known as *eventual consistency* [9]: after an update completes on a chunk, the system does not guarantee that subsequent accesses will return the updated value; there is an *inconsistency window* that represents the time period between an update and the moment in which any observer will see the updated value. To this end, all operations performed on chunk replicas must be *serialized*, i.e., a total order among operations must be defined to provide to each cloud node managing a replica the same ordered sequence of operations.

Achieving scalability requires also asynchronous communication among cloud nodes. To this end, the publish/subscribe paradigm represents an appealing solution due to its intrinsic decoupling properties in terms of time, space and synchronization [7]. Several publish/subscribe systems, in fact, are currently used to propagate updates across data centers worldwide, such as Google's Thialfi [1] and Yahoo!'s PNUTS [6]. In addition, this paradigm is also intrinsically characterized by a one-way information flow, from publishers to subscribers, so to avoid feedbacks that would create throughput oscillations in the system [4].

A mechanism for enforcing ordered notifications in topic-based publish/subscribe systems is presented in [2], and can be used as a building block for the development of an eventual consistency algorithm. The proposed solution relies on a set of Topic Managers (one per topic) interconnected in a direct acyclic graph (DAG) that act as timestamp generators for events published in the system. Each time an event is published on a topic T, the associated TM is contacted to start a *collaborative* timestamp generation procedure that involves the TMs of all topics T' till the DAG root, such that there are at least two subscriptions including both T and T'. Each TM attaches to the timestamp a sequence number that represents the number of events published on the topic it manages. Interested subscribers receive the event and the associated timestamp, that is used to infer the correct order of that event. In this way, two subscribers that receive the same two events can notify them in the same order (for further details see [2]).

The collaborative timestamp generation procedure has tree important characteristics:

- it generates *per-topic ad-hoc timestamps* that can be used to order only those events that could possibly be delivered out of order by distinct recipients. This solution, with respect to a simple total order of all the events injected in the system let us avoid the use of consensus primitives. In fact, it is well known that, even if consensus algorithms like Paxos are currently employed in cloud infrastructures (e.g. Google Ap-

LADIS '12 Madeira, Portugal

pEngine), they can perform unreliably over WAN links [3] and developers are thus under huge pressure to use them only if strictly necessary and for small-size groups of processes [5];

- it decreases the number of rollback operations done on replicated data with respect to a naive solution that notifies events as soon they arrive and then correct possible inconsistencies caused by a *wrong* delivery order. Typical rollback algorithms, in fact, require either the system stops processing updates or that live processes refrain from receiving new events, bringing to possible delays and discarded events that would produce unpredictable results to clients' queries;
- it defines a one-way timestamping path for each event, avoiding loops and feedbacks among TMs so to prevent throughput oscillations.

Starting from [2], we are currently studying how this solution can be improved for a cloud environment, that typically exhibits strong event production rates. Specifically, the goal is to efficiently allocate TMs on physical machines in order to reduce (i) network resource utilization, (ii) interaction among cloud nodes, and (iii) latency overhead during timestamp construction. In [2], in fact, no specific rule is given to allocate TMs on physical machines; as such, the generation of a timestamp may involve several machines (potentially as large the number of topics in the system), negatively affecting notification latency and network resource usage.

To achieve this goal, a trivial solution is represented by a single machine handling all TMs, that however would be easily overwhelmed by timestamp requests. A more efficient solution consists in a set of machines and a procedure that allocates TMs such that (i) each machine processes a bounded number of requests per time unit, and (ii) the information flow in input to the next machine in the DAG tends to zero.

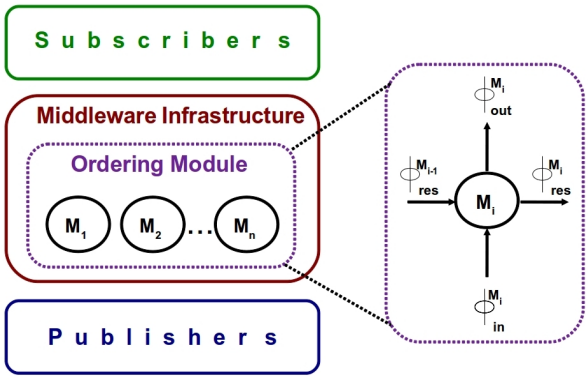

Figure 1: Architectural view of the solution in [2] with focus on the Ordering Module.

Figure 1 shows the internal architecture of the Ordering Module. Each physical machine M_i that constitutes the DAG has two input information flows: $\Phi_{in}^{M_i}$ from publishers and $\Phi_{res}^{M_{i-1}}$ from the machine M_{i-1}. In addition, M_i has two output information flows: $\Phi_{out}^{M_i}$ to subscribers and $\Phi_{res}^{M_i}$ to machine M_{i+1}.

The objective of our study is to design a distributed algorithm that maps TMs on physical machines so to minimize $\Phi_{res}^{M_i}$, with the constraint that each machine processes a bounded number of events per time unit. The mapping algorithm has to consider the interest of users (i.e., subscribers) to optimize subscriptions clustering, and then to assign a cluster to a single machine. In addition, the algorithm must also respect the relation among topics introduced by the DAG, so to preserve the one-way sequence of messages during the timestamp generation.

The ideal mapping of TMs on physical machines would generate $\Phi_{in}^{M_i} = \Phi_{out}^{M_i}$ and $\Phi_{res}^{M_i} = 0$, that is, all the traffic coming from publishers is locally processed by a machine M_i and then forwarded to subscribers, with no interaction with other machines. This means that each machine would work in isolation and that a single step would be required to ensure ordered notifications.

Cloud computing requirements impose to revise classical mechanisms to order events in a distributed systems like timestamping in a non-trivial way. Such mechanisms have to satisfy the CAP theorem and to avoid oscillation due to unexpected network load on some component while keeping short event delivery latency. This creates the condition for an exciting research agenda on innovative and efficient distributed architectures and solutions to classical coordination and communication distributed computing problems.

1. REFERENCES

[1] Atul Adya, Gregory Cooper, Daniel Myers, and Michael Piatek. Thialfi: a client notification service for internet-scale applications. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 129–142, New York, NY, USA, 2011. ACM.

[2] R. Baldoni, S. Bonomi, M. Platania, and L. Querzoni. Dynamic message ordering for topic-based publish/subscribe systems. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2012.

[3] Ryan Barrett. Transactions across datacenters. Google I/O developer conference `http://www.google.com/events/io/2009/sessions/TransactionsAcrossDatacenters.html`, 2009.

[4] K. Birman. Rethinking multicast for massive-scale platforms. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 1–1. IEEE, 2009.

[5] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[6] B.F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[7] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[8] R. Shoup. Architectural principles. `http://www.infoq.com/presentations/shoup-ebay-architectural-principles`.

[9] W. Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.