Language Expressiveness and Quality of Service for Publish/Subscribe Systems

Kaiwen Zhang∗
Middleware Systems Research Group
Department of Computer Science
University of Toronto, Canada
kzhang@cs.toronto.edu

ABSTRACT

The publish/subscribe paradigm is known for its loosely coupled interactions and event filtering capabilities. Traditional applications using pub/sub systems require large-scale deployment and high event throughput. Thus, pub/sub has always put the emphasis on scalability and performance, to the detriment of filtering expressiveness and quality of service. The matching language is usually limited to topic-based or content-based event filtering and does not allow complex stream-based subscriptions to be expressed. Messages are delivered on a best-effort basis without any ordering or reliability guarantees. Installing additional services and event processing systems at the endpoints can overcome the limitations of pub/sub systems. However, we argue that such solutions are inefficient and put a lot of strain on the pub/sub layer itself. Therefore, the focus of this thesis is to develop integrated solutions to extend pub/sub language expression and quality of service, and demonstrate that our approach results in better performance from a holistic perspective. We first describe multiple case studies for pub/sub and identify major features which need to be supported. We then extract those requirements from our use cases and develop general solutions within the pub/sub layer. Features we have supported so far includes total order and ranked data dissemination. Finally, we conduct experiments to compare the performance of our approach to baselines which rely on end-to-end services and perform holistic evaluations to assess the impact of our work.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Publish/Subscribe

General Terms
Algorithms, Design, Experimentation, Measurement, Performance, Reliability

∗PhD Adviser: Hans-Arno Jacobsen

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Middleware 2012 Doctoral Symposium, December 3, 2012, Montreal, Quebec, Canada.
Copyright 2012 ACM 978-1-4503-1611-8/12/12 ...$15.00.

Keywords
Publish/Subscribe, Complex Event Processing, Data Streams, Aggregation, Total Ordering, Online Games

1. INTRODUCTION

Publish/subscribe is a simple communication paradigm which allows loosely coupled entities to disseminate data in an event-based manner. Data producers publish data to a broker which forwards the publications through the broker overlay network to the interested data consumers. Interest in a publication is defined by consumers using subscriptions, to which each publication is matched against to determine the set of consumers to deliver to. Additionally, some pub/sub systems require the data producers to advertise the space they are publishing to initialize routing paths in the overlay network. Applications which employ the pub/sub abstraction include business process execution [25], workflow management [6], business activity monitoring [11], stock-market monitoring [26], selective information dissemination and RSS filtering [24], complex event processing for algorithmic trading [16], and network monitoring and management [11].

For the most part, these applications only require simple matching semantics. For instance, RSS filtering requires only topic-based matching, since each subscription is attached to a feed and all the publications generated within that channel are delivered to the matching subscriptions. Others can benefit from more advanced filtering capabilities which allow for predicates to filter within a topic (known as content-based matching). In the case of stock market monitoring, a client could define interest in a certain stock only if it reaches certain conditions (e.g., the value of the stock quote exceeds a certain threshold).

In order to satisfy the specifications of these applications, publish/subscribe systems have traditionally put the emphasis on high performance and scalability of a set of communication primitives with expressiveness limited to topic or content-based matching [10]. Recently, we have discovered applications which constitute a good fit for pub/sub due to its loosely coupled, many-to-many communication pattern, but require semantics which are not supported by current systems. For instance, current pub/sub systems are not capable of expressing interest computed over a window of publications. The ability to aggregate publications is necessary for applications such as sensor networks [2] or smart traffic monitoring [20], where large volumes of data are summarized and disseminated to the application layer.
These limitations can be alleviated by attaching a CEP (Complex Event Processing) system [18] at the subscribers (see Figure 1). Higher level subscriptions submitted to the CEP system at the client can be translated into pub/sub subscriptions: the interest of the original subscription can then be reconstructed by the CEP system using the publications received at the pub/sub layer. Figure 2 illustrates this approach. Coarse-grained subscriptions are thus submitted to the pub/sub system. The resulting stream of publications is then processed at the subscriber’s side to produce the final result.

High level expressions are usually not supported due to the overhead it puts on the pub/sub system. However, increasing the expressiveness of the subscription language can actually raise the scalability of the system. For instance, a content-based filter improves the selectivity of the subscriptions, which reduces publication traffic. We therefore argue that bridging the gap between the subscription space and the desired application interest at the endpoints is an inefficient solution and that we should strive to increase the expressiveness of pub/sub to reduce the amount of post-processing to be done at the subscriber’s end.

The goal of this thesis is to extend the subscription language for the purpose of improving the performance of the system. Typically, this involves reducing the amount of traffic by allowing the subscribers to specify their interest at a finer granularity.

A second objective is to investigate quality of service for pub/sub systems. While there is a body of work concerning fault-tolerance [22, 14], little work exist on other delivery guarantees such as ordering [32]. Again, QoS guarantees can be enforced via an external service deployed at the endpoints, but more efficient solutions can be achieved within the pub/sub system.

Finally, we also aim at providing internal optimizations for pub/sub systems. These optimizations arise from the properties of the studied workloads.

The rest of this document discusses the current and future work for the thesis. First, we describe online games as an use case and motivating scenario for pub/sub systems. Second, we describe total order and ranked data dissemination, our two most mature contributions. Finally, we describe aggregation and advertisement transformations which are more recent works still under investigation.

## 2. ONLINE GAMES

Massive Multiplayer Online Games (MMOGs) is a popular genre of virtual reality games played over the Internet. Players in the game control a character (avatar) within a shared world with other players in the game. Players have the ability to interact with each other and the environment, altering the state of the game by taking actions. This state is persisted and continuously evolve over time, as players are able to make progress with their character during each and every session [29].

Each player client renders a view of the world based on the location of the player’s character. In particular, game objects are typically replicated to the client’s machine for local access. Due to the large game space, players only maintain replicas of objects close to their character’s position (see Figure 3: the yellow player’s interest radius is bounded by the red circle). Whenever the state of an object changes, the update must be disseminated to all clients holding a replica of the object and applied accordingly. Since characters can change position during the game (through movement), the process of dynamically maintaining the set of interested objects is called Interest Management [3].

Publish/subscribe is a particularly attractive choice to disseminate game state updates to the players. The pub/sub filtering capabilities are necessary to reduce the amount of traffic required for disseminating relevant updates to each client. Furthermore, we can also use content-based publish to perform location-based interest management. Players can...
subscribe to the radius around their current position. Game updates are localized and disseminated as publications with a position value.

Due to the interactive nature of MMOGs, disseminated updates can be either conflicting or non-commutative [28]. For instance, two players may be attempting to pick up the same object, or a player may attempt to purchase an item while its price is increasing. In both cases, the final outcome depends on the ordering of the updates. Thus, to prevent inconsistencies from occurring between different clients’ view, we require the pub/sub system to enforce a total ordering on the update publications.

Another issue stems from the nature of the game workloads. The distribution of players across the game world is not uniform. Gameplay is often driven by time-limited events which occur at specific locations. Players have a tendency to converge towards points of interest in the world; a phenomenon known as flocking [5]. This means that the rate of interaction increases at an exponential rate, since clusters of players form, where each player acts both as a data source and sink to the surrounding players contained in the hot spot. To curb these bursts, we introduce the notion of ranked data to allow player clients to prioritize and limit the updates they receive through the pub/sub system.

Finally, advertisements and subscriptions have to be reissued every time a player changes position. As a consequence, workloads contain a high number of advertisements and subscriptions. Current pub-sub systems are tuned for publication-heavy workloads, hence new techniques must be employed to deal with game workloads.

For instance, parameterization allows dynamic values in subscriptions [12]. Players only need to issue a single subscription centered around their player location while updating this subscription with every position change. However this work still requires the subscribers to manually update their subscriptions on a regular basis. Our work on advertisement and subscription transformations seeks to reduce the overhead of advertising and subscribing by adaptively tuning ads/subs at each broker in the pub/sub system which obviates the need of disseminating new ads/subs.

We will evaluate our approach by developing a pub/sub engine for Mammoth [15], a MMOG research framework. The current network engine is a centralized pub/sub engine which has been proven to be the bottleneck of system, limiting the scalability with respect to number of connected clients [8]. We plan to replace this engine with a distributed overlay-based pub/sub network and apply our enhancements there.

3. THESIS CONTRIBUTIONS

We list the different contributions of the thesis, detailing the current and future work.

3.1 Total Order

Total order ensures uniform publication delivery across subscribers [7]. It is useful in application scenarios where multiple subscribers can synchronize or observe their state through an external mechanism. As explained above, total ordering is necessary in online games in order for different clients to observe the same game state. Game state updates must be applied in an uniform order across the clients [15]. It is also used for fairness in competitive environments, such as stock markets [27].

The exact semantics we are using is called Pairwise Total Order and is defined as follows:

- **(Pairwise Total Order)** For any two publications \( p \) and \( q \), which are delivered to both \( s_1 \) and \( s_2 \), \( p \) is delivered using paths \( P_1 \) and \( P_2 \) to \( s_1 \) and \( s_2 \), respectively. Similarly \( q \) is delivered using paths \( Q_1 \) and \( Q_2 \) to \( s_1 \) and \( s_2 \), respectively. If \( P_1 \cap P_2 \cap Q_1 \cap Q_2 \neq \emptyset \), then FIFO links preserve pairwise total ordering between \( p \) and \( q \) for \( s_1, s_2 \).

Total ordering is enforced when the delivery paths between pairs of publishers and subscribers have a common node. Brokers can locally verify (without global knowledge) whether the natural ordering will take effect or not; if not, a special resolution protocol is used to determine the ordering of conflicting publications. Our algorithm is therefore able to contain the resolution overhead only to conflicting publications at conflicting subscribers. Figure 4 illustrates how the algorithm function. A publication is sent by publisher P1 to broker B1 and needs to be delivered to subscribers 1-4. Concurrently, a publication is sent by P2 to broker B2 for subscribers 1-3. According to our definition of pairwise total order, subscribers 1-3 must deliver publications from P1 and P2 in the same order. B1 engages in a detection step and determines that the path P1-S1 does not share a common broker with P2-S2 and P2-S3. Therefore, the natural total order property does not hold and B1 needs to trigger a resolution phase. Notice that S4 is unaffected by this conflict and receives the publication right away. Likewise, the publication by P2 triggers resolution at B2 as well. During the resolution phase, B3 will also be involved in the resolution phase to handle the delivery of the publications to S3. Once the resolution phase is over, brokers B1-3 will deliver the publications in the correct order to S1-3, respectively.

Future work in this direction includes support for cyclic topologies and employing techniques such as client mobility and topology reconfiguration to minimize the need for conflict resolution.
this functionality to top-k publications most relevant to the subscriber. We also call the top-k score of each publication and only disseminates the publications based or time-based, sliding or tumbling [1]). Compute the according according to the given window semantics (count-based windows which cut across broker boundaries.

The main challenge of the work lies in maintaining correctness of the top-k results in count-based windows. We are instead considering dissemination or distribution of the top-k processing [31]. Our approach seeks to apply the top-k filter as close to the sources as possible in order to reduce traffic within the pub/sub network (see Figure 5). Top-k results from multiple sources are then collected and merged downstream to produce a single top-k stream for the subscriber. The related work thus serves as the building blocks for our solution, which focuses on distributing the top-k processing to reduce the publication traffic in the system. The main challenge of the work lies in maintaining correctness of the top-k results in count-based windows. We establish a correctness criteria which is based on the concept of stream constructibility:

**Theorem 2. (Stream Constructibility)** Given a set of delivery guarantees for a given publish/subscribe system, a stream of top-k publications T for subscription S is considered correct if there exists an interleaving of all matching publications for S, which is constructed according to the delivery guarantees, which can be processed according to the top-k semantics to obtain T.

The idea is that any top-k results is obtainable from expanding the results into some possible initial stream of input publications and applying top-k. In our case, we only assume that publications follow FIFO order on a per-publisher basis. We are therefore allowed to change the order of publications from different sources when verifying correctness.

In a naive early filtering solution, source brokers simply filter local publications and select the top publications out of each window to disseminate downstream. Non-source brokers then simply forward any received publications to the subscriber. Figure 6 is a sample execution of a naive top-k algorithm for sliding count-based windows. Each source broker independently select and forward the top-k publications out of their respective windows. For instance, the top broker forwards publications [a, d] for the first window and [b] only for the second window (publication d is selected again but already disseminated: it is omitted). The top-k results from each source broker are delivered in a certain interleaving to the subscriber broker.

Suppose the order is [a, d][2, 3][b][4, 5]. In a centralized solution, where every publication is first sent to the subscriber, which then perform top-k calculations, such interleaving of top-k results can be derived from a reconstructed stream of original publications as [a, b, c, d, 1, 2, 3, 4, b, c, d, e, 2, 3, 4, 5]. However, applying our sliding window semantics to such stream would also require results for windows such as [b, c, d, 1] and [2, 3, 4, 6]. In a naive distributed solution, such windows are not considered since each source broker only consider windows of publications originating from their own publishers. In other words, this simple solution fails to consider windows which cut across broker boundaries.

To address this issue, we introduce a distributed solution for disseminating top-k publications based on the novel notion of chunks and guards. We show that by adopting a hybrid strategy where publisher brokers switch between forwarding of entire windows of publications and forwarding of processed top-k results, a sufficient amount of data is disseminated downstream for subscriber edge brokers to be able to merge those streams and produce correct top-k results.

In our future work, we plan on optimizing our solution in light of the sensitivity analysis we have performed. In particular, the throughput benefits observed drop dramatically as the number of subscribers increase. Synchronizing win-
dows across subscriptions will improve performance in that situation.

3.3 Ads/Subs Transformations

We would like to allow the brokers to transform existing advertisements and subscriptions to improve the performance of the system. The changes are internal to the pub/sub layer and are opaque to the end-users.

This work exploits the notion of subscription covering [4]:

- (Subscription Covering) A subscription $S$ covers subscription $T$ if $\forall P \in T, P \in S$, where $X$ is a publication.

In other words, any possible publication matching $T$ will also match $S$, since the subscription space of $S$ is a superset of $T$. If a broker propagates a subscription $S$ and later receives a subscription $T$ which is covered by $S$, then the broker does not need to propagate $T$ any further since it is already receiving all publications matching $T$ as they match $S$ as well.

This optimization allows the broker to save propagation cost.

The same concept applies for covering advertisements.

Currently, covering is used passively: brokers can stop the propagation of an ad/sub only if a previous advertisement covers it. Our goal is to pre-emptively enlarge ads and subs such that future ads/subs are covered, thus reducing traffic (a process we call aggressive merging). This is particularly effective for advertisements since they are flooded in the network according to the advertisement-based forwarding model, and is the primary focus of our work.

The benefits of aggressive merging are speculative. Enlarged advertisements will match more subscriptions than necessary. If the enlarged space is larger than the future advertisements, subscriptions may be falsely matched and their propagation becomes unnecessary. We therefore develop a cost model which considers the savings from the covering provided by the transformed advertisement and the cost of propagating false positive (i.e. attracting subscriptions which not matched to any real advertisement) subscriptions.

We also develop techniques for estimating the various components of the cost model. For instance, brokers can locally maintain a time series of submitted advertisements and extrapolate how the aggregated advertisement space will grow. Merging heuristics, such as closing gaps in between contiguous ranges [17], can be used to minimize the false positive rate.

3.4 Distributed Aggregation

Certain use cases require large quantity of publications to be aggregated before delivering to the subscriber [21,2]. For instance, a traffic monitoring service may be more interested in the average speed of vehicles passing by a certain location rather than the raw data collected for every vehicle. Although subscribers could subscribe to the entire stream and aggregate it locally, the throughput is not sufficient to support the streaming rate. We want to integrate the aggregation computation to the pub/sub layer, such that the aggregation is computed as close as possible to the sources, an approach similar to the one used for top-k publications.

We have developed and implemented a distributed aggregation solution for time-based windows in pub/sub. The solution relies on early aggregation at the source brokers of local publications. These partial results are then collected at each intermediary broker and merged before propagating downstream. The solution supports a wide range of operators with various distribution properties [13]. We also employ various optimization techniques to exploit overlaps between aggregation subscriptions and batching techniques to minimize inter-broker traffic.

One open problem is dealing with aggregation subscriptions over very short windows. In such cases, it is possible that the rate of publication is lower than the rate of aggregation. It is then more efficient to forward all publications rather than computing and notifying aggregation results over empty windows. We aim to develop an adaptive solution for switching between publication and aggregation forwarding.

4. SUMMARY

The primary focus of pub/sub systems is performance. Expressiveness and Quality of Service can be addressed at the endpoints using external services. The goal of this thesis is to leverage additional expressive or QoS features within the pub/sub layer to improve the performance of the entire system. Using massively multiplayer online games as a study case, we extracted several key features and enhancements which need to be supported by pub/sub systems.

We extend the language to increase the selectivity of the subscriptions, which reduces traffic consumption, as demonstrated by our top-k and aggregation work.

The total order work exploits the pub/sub topology to efficiently enforce an uniform ordering with contained resolution overhead.

Finally, we also develop internal optimizations for pub/sub which reduces the cost of ads/subs by transforming them within the brokers to maximize covering.

5. REFERENCES


