Pub/Sub on Stream: A Multi-Core Based Message Broker with QoS support

Zhaoran Wang\textsuperscript{1}, Yu Zhang\textsuperscript{2}, Xiaotao Chang\textsuperscript{2}, Xiang Mi\textsuperscript{1}, Yu Wang\textsuperscript{1}, Kun Wang\textsuperscript{2}, Huazhong Yang\textsuperscript{1}

1. Dept. of Electronic Engineering, TNList, Tsinghua University  
2. IBM Research - China
Motivation

- Publish/Subscribe (Pub/Sub) is becoming an increasingly popular message delivery technique in the Internet of Things (IoT) era.
  - Smart Grid
  - Transportation
  - Battlefield Monitoring

- However, classical Publish/Subscribe is not suitable for some emerging IoT applications due to its lack of QoS capability.
  - Smart Grid: late messages $\Rightarrow$ severe power grid failure

- **Goal**: Enable QoS support in a Publish/Subscribe message broker with a multi-core processor.
Contents

■ Motivation
■ QoS problems in Pub/Sub Systems for IoT applications
■ Analysis and Partitioning of the Sequential Matching Algorithm
■ Stream Matching Framework
  ▸ Basic Framework
  ▸ Deadline-aware Fine-Grained Scheduling (DFGS)
  ▸ Smart Dispatch
■ Evaluation
Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling (DFGS)
  - Smart Dispatch
- Evaluation
QoS Problems in Pub/Sub Systems

- Publish/Subscribe (Pub/Sub) systems

![Diagram showing the relationship between publishers, subscribers, and a message broker. Publishers send messages to the broker, which then routes them to subscribers based on their subscriptions.]
QoS Problems in Pub/Sub Systems

- Applications such as the Smart Grid introduce various requirements on QoS.

Table 1: Normalized Values of QoS+ Parameters

<table>
<thead>
<tr>
<th>Difficulty (5 hardest)</th>
<th>Latency (ms)</th>
<th>Rate (Hz)</th>
<th>Criticality</th>
<th>Quantity</th>
<th>Geography</th>
<th>Deadline (for bulk traffic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5–20</td>
<td>240–720+</td>
<td>Ultra</td>
<td>Very High</td>
<td>Across grid or multiple ISOs</td>
<td>&lt;5 seconds</td>
</tr>
<tr>
<td>4</td>
<td>20–50</td>
<td>120–240</td>
<td>High</td>
<td>High</td>
<td>Within an ISO/ RTO</td>
<td>1 minute</td>
</tr>
<tr>
<td>3</td>
<td>50–100</td>
<td>30–120</td>
<td>Medium</td>
<td>Medium</td>
<td>Between a few utilities</td>
<td>1 hour</td>
</tr>
<tr>
<td>2</td>
<td>100–1000</td>
<td>1–30</td>
<td>Low</td>
<td>Low</td>
<td>Within a single utility</td>
<td>1 day</td>
</tr>
<tr>
<td>1</td>
<td>&gt;1000</td>
<td>&lt;1</td>
<td>Very Low</td>
<td>Very Low (serial)</td>
<td>Within a substation</td>
<td>&gt;1 day</td>
</tr>
</tbody>
</table>

- Tight latency requirements for critical messages.

[Dave Bakken, Proceeding of IEEE 2011]
QoS Problems in Pub/Sub Systems

- Failure may lead to severe results.
  - Inefficient feedback control
QoS Problems in Pub/Sub Systems

- Messages with the same QoS requirements ➞ Channel
- A message's QoS requirement, especially latency, is violated ➞ Failure
QoS Problems in Pub/Sub Systems

- Inappropriate allocation of resources will lead to violation of QoS requirements.
QoS Problems in Pub/Sub Systems

- **Solution 1:** Resolve the QoS violation by splitting and scheduling.

![Diagram](image.png)

- **Channel 1**
  - Message 1's deadline
  - Message 2's deadline

- **Channel 2**
  - Message 3's deadline

- **Channel 3**
  - Message 5's deadline

- **Core 1**
  - Message 1's deadline
  - Message 3's deadline
  - Message 2's deadline
  - Message 5's deadline
QoS Problems in Pub/Sub Systems

- **Solution 2:** Resolve the QoS violation by dispatching.

Channel 1
- Message 1's deadline
- Message 2's deadline

Channel 2
- Message 3's deadline
- Message 5's deadline

Channel 3

Core 1
- Message 1's deadline
- Message 3's deadline
- Message 5's deadline

Core 2
- Message 2's deadline
- Message 5's deadline

Time
Key Idea

- **Key idea**: actively schedule computation resources to guarantee QoS requirements of messages.
  - Abstract the message matching algorithm into a task-based framework.
  - Two-level task scheduling mechanism to allocate the computation resources
    - Intra-core fine-grained task scheduling
      - Deadline-aware Fine-grained Scheduling (DFGS)
    - Inter-core message dispatching
      - Smart Dispatch
Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling (DFGS)
  - Smart Dispatch
- Evaluation
Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)

<table>
<thead>
<tr>
<th>Event</th>
<th>p0</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to get</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>entrance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**H phase**

- Set bit vector

- Hash to get entrance

**C phase**

- Sub Cluster
- Sub Cluster
- Sub Cluster
- Sub Cluster
- ...
Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)

<table>
<thead>
<tr>
<th>Number of Subscriptions</th>
<th>Processing time/ms</th>
<th>H Phase</th>
<th>C Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>0.034</td>
<td>65%</td>
<td>35%</td>
</tr>
<tr>
<td>60,000</td>
<td>0.15</td>
<td>14%</td>
<td>86%</td>
</tr>
<tr>
<td>600,000</td>
<td>0.81</td>
<td>3%</td>
<td>97%</td>
</tr>
</tbody>
</table>
Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)

**H phase**

- Event
  - p0
  - p1
  - p2
  - p3

- Hash to get entrance

- Set bit vector

- Attribute1
  - =
  - >
  - <
  - <=

- Attribute2
  - =
  - >
  - <
  - <=

- Attribute3
  - =
  - >
  - <
  - <=

**SubC phase**

- AP1
- AP2
- AP3
- AP4
- ...

**C phase**

- Sub Cluster
- Sub Cluster
- Sub Cluster
- Sub Cluster
Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling (DFGS)
  - Smart Dispatch
- Evaluation
Stream Matching Framework

- The architecture of the task-based framework
Stream Matching Framework

- **How tasks are executed on each core.**

**Worker Thread:**

```c
While(1){
    Task=pop(Task_Scheduling_Queue);
    execute(Task);
}
```

**Task Execution:**

```c
Data=ReadInput(Input_Buffer);
Result=Process(Data);
WriteOutput(Output_Buffer, Result);
```

**Legend**

- Tasks for channel 1
- Tasks for channel 2

---

**Worker Thread of Core 1**

- H Phase
- AP Splitter
- SubC Phase_1

**Worker Thread of Core 2**

- SubC Phase_1
- H Phase

**Core k**

- Task scheduling queues

**Core k+1**

- Task popped and executed

**Splitter inserts tasks**
Motivation and the Key Idea

QoS problems in Pub/Sub Systems for IoT applications

Analysis and Partitioning of the Sequential Matching Algorithm

Stream Matching Framework
- Basic Framework
- Deadline-aware Fine-Grained Scheduling (DFGS)
- Smart Dispatch

Evaluation
Stream Matching Framework

- Deadline-aware Fine-Grained Scheduling (DFGS)
  - Splitter decides where in the task scheduling queue the tasks for a message should be inserted.

  - 3 criterions for inserting a task T:
    - T won’t cause any task after it to fail
    - Intra-channel order is preserved
    - T's deadline won’t to be violated

  - Linearly traverse the task scheduling queue to find the position satisfying the 3 criterions
    - Return false if there isn’t any.
    - Some optimizations in the paper.
Stream Matching Framework

**Pros**

- The tasks are **NOT** executed in the order the message comes.
- Urgent message get processed first.
- Urgent message can interrupt the processing of normal messages.
Stream Matching Framework

- **Smart Dispatch**
  - Splitter polls to decide which way to dispatch a message
    - Check resources on each core using DFGS
  - If All cores returns false, message is deserted and counted as a failure
Stream Matching Framework

- **Pros**
  - Explores the parallelism of multi-core machines by resource checking globally

![Diagram showing message deadlines and processing on two cores](image_url)
Stream Matching Framework

- Cons of DFGS and Smart Dispatch
  - Scheduling overhead
    - Linear complexity for each core
    - The number of cores is limited
Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling (DFGS)
  - Smart Dispatch
- Evaluation
Evaluation

- **Settings**
  - Intel Xeon CPU E5507 at 2.27GHz
  - Fedora with kernel version 2.6.31.5
  - GCC 4.1.1 using -O3
  - **Dataset**
    - 1504 predicates; 238 attributes; Channel 1 with 60,000 subscriptions; Channel 2 with 600,000 subscriptions as in [F. Fabret, SIGMOD 2001]
    - Set the message processing deadline according to [Dave Bakken, Proceeding of IEEE 2011]
Evaluation

- How Parameters affects the failure rate?
  - QoS-ignorant system as the baseline
  - 2 channel for the convenience of discussion
  - Failure rate:
    - QoS-aware system, the message fails to be dispatched because there are no resources
    - QoS-aware or QoS-ignorant systems, the message's deadline violated
    - The percentage of total failed messages in all messages

- Scalability on multi-core machine?
Evaluation

- The processing time's effects on the failure rate
Evaluation

- The event’s arriving period’s effects on the failure rate

![Graphs showing failure rate vs. rate of channel 2 with 1 Core and 7 Core, comparing QoS-ignorant and QoS-aware models.](image-url)
Evaluation

- The event’s arriving period’s effects on the failure rate
  - Smart Dispatch or DFGS?

Channel 2 has shorter processing time  
Channel 2 has longer processing time
Evaluation

- Resources’ effects on failure rate

![Graph showing the effects of resources on failure rate. The graph compares QoS-ignorant and QoS-aware systems. The x-axis represents the number of cores used, ranging from 0 to 8. The y-axis represents failure rate, ranging from 0% to 100%. The graph shows a significant reduction in failure rate as the number of cores increases for both systems, with the QoS-aware system consistently lower than the QoS-ignorant system.]
Evaluation

- Maximum scalability

![Graph showing scalability comparison between QoS-aware and QoS-ignorant systems](image-url)
Conclusion

- Enable QoS support in a Publish/Subscribe message broker with a multi-core processor
  - Intra-core fine-grained scheduling
  - Inter-core message dispatching mechanism to provide

- Discuss about how parameters affect the message failure rate
  - Period
  - Processing time
  - The number of cores used

- Near-linear scalability/10x maximum throughput
THANK YOU😊