Resilient Delay Sensitive Load Management in Environment Crisis Messaging Systems

Ran Tao, Stefan Poslad, John Bigham
Dept. of Electronic Engineering and Computer Science
Queen Mary, University of London
London, UK
{ran.tao, stefan.poslad, john.bigham}@eecs.qmul.ac.uk

***Abstract*—Typical environment crisis messaging systems, e.g., those used in Tsunami Early Warning Systems, are open, distributed, and heterogeneous. In such systems, Publish Subscribe Message Oriented Middleware (PSMOM) is widely deployed using message brokers to enable open and distributed data publishers and subscribers to exchange raw and processed sensor data, authority driven workflows, and information generated by citizens. A key security challenge is that such message brokers may suffer a Denial of Service (DoS) attack, becoming overloaded and resulting in performance degradation or even worse in a broker crash. This significantly decreases the effectiveness of the system as vital messages may face unexpected delays or become lost. In order to address this challenge, a resilient workload management framework is required to better redistribute the message exchange from overloaded brokers to brokers with lesser loads. However, existing workload management mechanisms are not suitable to manage load in such environment crisis messaging systems as they are not designed to handle message traffic that may have different Quality of Service (QoS) requirements, e.g., different end-to-end transmission latency requirements. These may cause unexpected delays for sensitive messages or trigger unnecessary load balancing. In this paper, we propose a resilient delay sensitive workload management framework that extends an existing state-of-the-art messaging system, Publish/Subscribe Efficient Event Routing (PEER), by adding support for workload allocation, a Queue Depth load metric, and dynamic load thresholds, enabling end-to-end latency guarantees and avoiding unnecessary load balancing. The model has been validated in a simulation.**

Keywords-PSMOM; Denial of Service attack; Workload Management

I. INTRODUCTION

Modern environment crisis management systems, such as Tsunami Early Warning Systems (EWS) follow a System-of-System (SoS) framework that integrates various messaging components and subsystems, e.g., different information sources, processing services, and crisis simulation systems, and takes into account the open, distributed, heterogeneous, and collaborative nature of such systems. In such a SoS framework, PSMOM is deployed as a messaging bus because it allows components and subsystems to be distributed on heterogeneous platforms and to communicate asynchronously in a loosely coupled manner [1]. In addition, QoS-aware policies can be used to help differentiate message traffic in a PSMOM to allow different types of data, such as raw and processed sensor data, service data, and simulation data to be exchanged via inter-linked message brokers [13]. Figure 1 shows an example EWS framework based on PSMOM (Messaging Bus) support. In this framework, “P” are message publishers or pubs that label messages with respect to different subjects and send these to message Brokers “B”. “S” are subscribers or subs that request the messages of interest to them and receive messages matched to their interests via a message broker. The message interaction in the system consists of the following. First, a sensor data bus type broker acquires physical sensor data from different sources, e.g., physical sensors, such as buoys and tide gauges, and human sensed data via social networks data sources, such as Twitter on mobile phones. Second, a database (DB) receives and records the live sensor data and publishes the historical sensor data via a processing message broker or bus. Third, this processing bus receives both live and historical sensor data, processes this data and publishes the analysis results to a User Interface (UI). Fourth, the UI receives and displays the analysis results. Fifth, a service controller publishes service control messages to message components when they need to change its performance to adapt to a changing environment situation, e.g., to increase the sensor data collection frequency in case the onset of a crisis is detected. With the support of PSMOM, these system components can be distributed in monitor centres at different geographically locations and work collaboratively.

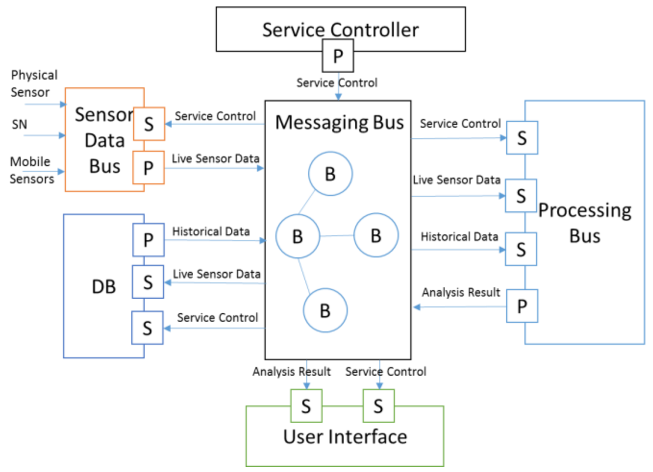

Figure 1. EWS with PSMOM Support

A core security risk in such environment crisis messaging systems is a Denial of Service attack [13] that significantly reduces the efficiency and accuracy of an early warning when message brokers become overloaded. This can be caused by: rogue publishers that can flood the broker with large fake messages, high-rate messages, and many useless topics; rogue subscribers with a slow subscription speed can cause messages to build up in the broker. A standard method to avoid such problem is to use user authorization, i.e., only authorized pubs and subs are legal and able to exchange messages using the message broker. However, this blocks the unauthorized publishers that could come online and provide useful information to improve the ground truth at a crisis. The above attacks can be modeled as a message burst (rogue publisher attack) and a capacity reduction (rogue subscriber attack). Workload management through an improved broker resilience model, e.g., mirroring and load balancing, is a feasible solution. Some forms of resilience, such as mirroring, are quite standard and are already supported in our resilient messaging system. Instead, in this paper, we focus on a more challenging workload management sub-system to provide load balancing for message brokers in EWS.

Existing MOM workload management mechanisms are not applicable in EWS because of the following limitations. First, much work focuses on homogeneous broker models where brokers are assumed to have the same processing power and bandwidth. However, EWSs tend to be heterogeneous because different system components and subsystems have varying CPU, memory, disk size and network bandwidth. Second, the heterogeneity of messages is not fully considered. Although messages have been divided into different subjects and assigned with different sizes and rates, different QoS requirements for different types of messages are ignored. This may trigger unnecessary load balancing and result in a waste of system resources or introduce unexpected delays to time-critical messages and result in a delay for critical decision-making.

In this paper, we propose a delay sensitive workload management solution for PSMOM used in EWS. This solution extends the Publish/Subscribe Efficient Event Routing (PEER) framework [1] by adding a workload distribution mechanism that assigns message brokers with least utilized load capacities to clients, a Queue Depth load metric and dynamic thresholds, to provide latency guarantees and to avoid unnecessary load balancing.

The remainder of the paper is organized as follows. Section II describes related work. Section III shows the system overview. Section IV describes the workload management framework. Section V presents a validation of the framework. Section VI reports the conclusions and projects the future work.

II. RELATED WORK

Load balancing in distributed system has been widely researched for over two decades [1, 4]. The goal of load-balancing solutions is to efficiently distribute the workload to the available resources so as to lower the risk of system overload and to maintain system performance.

Load balancing solutions can be executed in different layers: the network layer, operating system layer, middleware layer, and application layer. The layer, where the load balancing mechanisms can effectively detect and balance the load, is the best place to deploy the solution. For example, it would be ineffective to use a random DNS redirection strategy in the network layer or perform process migration in the OS layer for load balancing. This is because these approaches cannot identify the relationship between subscriptions nor estimate the load imposed by a subscription onto a broker [1]. Therefore, we focus on the load balancing strategy in the middleware layer as a PSMOM system is middleware based.

In a PSMOM system, the broker workload depends on the number and type of subscriptions served by this broker, i.e., on message size and the incoming and outgoing messages rates. Load balancing in a PSMOM is achieved by migrating subscriptions from overloaded brokers to ones with lesser loads.

Gupta et al. [6] proposed two types of load balancing in a peer-to-peer content-based PS system [2, 12]. Load balancing is achieved by splitting the peer with the heaviest subscription load in half and propagating events to a newly joint replicated peer. Chen and Schwan [7] proposed an optimized overlay reconstruction algorithm that performs load distribution based on CPU load. Load Balancing is triggered only when clients find a broker that is closer than its current connected broker. Subscription clustering [8, 9, 10, 11] is another solution that partitions a set of subscriptions into a number of clusters in order to reduce the overall network traffic. The above solutions can balance the load but they are all designed for homogeneous systems.

Cheung et al. [1] proposed the PEER framework that aims to overcome the above limitations for load balancing in PSMOM. Its primary target is content-based PSMOM but the author claims that it can also be applied to topic-based PSMOM. In PEER, brokers have different processing capabilities and Internet links. The load of a broker is detected by periodically monitoring three middleware layer load metrics: input utilization, matching delay, and output utilization, and comparing the monitoring results of each metric with two static thresholds. Among these metrics, input utilization is determined by the quotient of the input rate (R_{input}) in messages per second over the matching rate ($R_{matching}$) in messages per second, i.e., $R_{input}/R_{matching}$; matching delay is defined as the average time (in second) spent in a broker to process matching; output utilization is defined as the quotient of the used bandwidth (BW_{used}) over the total bandwidth (BW_{total}), i.e., BW_{used}/BW_{total}. If unbalanced load or overload is detected, a load balancing is triggered and the system migrate subscriptions from the offloading broker onto a load-accepting broker, while not overloading it. An evaluation of the design compared to a naive random load balancing approach shows that PEER is capable of efficiently balancing load in a heterogeneous messaging environment. However, PEER ignores the heterogeneity of system applications, such as the different end-to-end latency requirements, and therefore may trigger unnecessary load balancing if all the applications are delay tolerant or introduce unexpected delays for delay sensitive applications. In addition, it does not distinguish the uplink that is used to disseminate messages to subscribers and downlink that is used to receive messages. The differences between these may introduce different client migration priorities in a load-balancing phase. Further, there is no pre-emptive workload distribution mechanism in PEER. It therefore requires extra work to migrate subscriber clients from one (edge) broker to another based on the load differences.

Our work extends the PEER framework by adding support for workload allocation and more comprehensive delay sensitive aware load detection, and redesigns the load analysis and balancing mechanisms to fit the detection and distribution mechanism.

III. System Overview

In Figure 1, multiple message brokers (B) form a messaging bus that works as an integrated message exchange. In our design, these brokers are organized into a Head-Edge Broker model that is motivated by the architecture adopted by Google's distributed publish/subscribe system GooPS for use in MOM deployments in real world applications [1]. Our design targets enhancements to the Head-Edge Broker model (Section III.A) by providing delay sensitive load management supported with management agents (Section III.B).

A. *Head-Edge Broker Model*

The Head-Edge broker model (H-E model) organizes the brokers into a hierarchy structure, as shown in Figure 2.

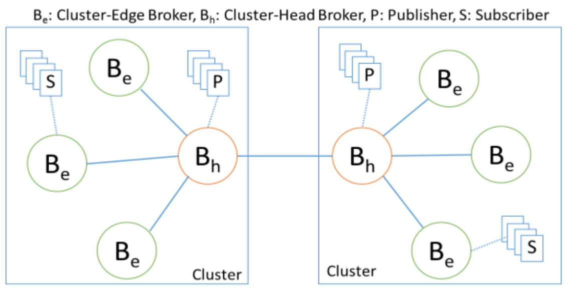
B_e: Cluster-Edge Broker, B_h: Cluster-Head Broker, P: Publisher, S: Subscriber
S
B_e
P
B_e
B_h
B_e
Cluster
P
B_e
B_h
B_e
B_e
S
Cluster

Figure 2. PEER Head-Edge Broker Model

A broker with more than one neighbour broker is referred to as a cluster-head broker (B_h), while a broker with only one neighbour broker is referred to as a cluster-edge broker (B_e). A cluster-head broker together with its connected edge brokers form a cluster. In the H-E model, publishers are only served by B_h, and subscribers are only served by B_e, so that, in a cluster, messages are always routed from the B_h to B_e. Inter-cluster message dissemination is achieved by having a B_h forwarding publication messages to the B_h of all matching clusters.

B. *Management Agent*

To manage the workload for H-E model, a Management Agent (MA) is allocated for each broker. The MA belonging to the head broker is called HMA, while the one belonging to the edge broker is named EMA. Both HMA and EMA consist of an Overlay Manager (HOM and EOM respectively), a Load Detector (HLD and ELD), and a Load Analyser (HLA and ELA).

HOM receives the broker allocation request from all the clients in the cluster and assigns brokers to the clients according to the client's source (a publisher or a subscriber), the availability of the broker, and the distribution status of existing clients. In addition, when load balancing is triggered, HOM notifies selected clients to migrate from original brokers to the new load-accepting brokers. What's more, HOM interacts with EMA to update the load information of edge brokers, and interacts with HOM of other clusters to share the cluster-based load information. ***EOM*** updates the load status of the edge broker to HOM and receives the load status of other edge brokers in the same cluster from HOM. In addition, when load balancing is triggered, EOM updates the available selected subscriptions to the HOM. Both HOM and EOM work with its relevant load analysers to generate an offloading client list that contains the clients to be migrated from the overloaded broker to the load-accepting broker when load balancing is required.

HLD and ***ELD*** detect the load status, e.g., as a set of fuzzy states, LOW, HIGH, and OVERLOAD, of the relevant broker, i.e., HLD monitors the head broker and ELD monitors edge brokers. Although the authors in [1] claim that the head broker is less likely to be overloaded since it does no matching work for subscribers, the head broker can become overloaded when it reaches its maximum network capacity whilst exchanging messages. So, HLD monitors the network bandwidth used by the head broker and reports its status to HOM when its load state changes. ELD does similar work but it needs to monitor all the load metrics (see section IV.B) and report this to the EOM. In addition, to get the dynamic threshold, ELD periodically detects the transmission latency between the edge broker and head broker, between subscribers and edge brokers, and request HLD to detect the transmission latency between publishers and head brokers.

HLA and ***ELA*** analyse the load distribution for clients, e.g., the Internet usage of individual client, store the observations into a table and pass this to the relevant OM. In addition, the clients in the overloaded broker are prioritized for offloading when its load metric exceeds its threshold otherwise making the broker become overloaded.

IV. Load MAnagement Framework

In this design, the workload management framework consists of a workload distribution phase, a load detection phase, and a load-balancing phase. In the workload distribution phase, HOM allocates brokers to each new subscriber based on the load status of the edge brokers and the distribution of existing subscribers. In the load detection phase, the load of the broker is periodically detected and the change of the load status is updated and sent to its OM. During the load balancing phase, a three step offloading strategy is adopted, i.e., locating the load-accepting broker(s), selecting subscriptions, and migrating the selected subscriptions from the overloaded broker to the load-accepting broker(s).

A. *Workload Distribution*

In practice, it is very important to avoid the OVERLOAD problem by optimizing the workload distribution beforehand. In this workload management framework, the workload distribution process is designed using the following principles:

- First, subscribers of the same topic are allocated to the same broker to avoid extra network bandwidth usage, as same messages are no longer routed to different edge brokers.
- Second, topics that are highly correlated are allocated to different brokers [5], as they may introduce a sudden increase in broker load.
- Third, new subscriber clients are allocated to brokers that have the least utilized load capacity that is computed from all the load metrics (Section IV.B).

B. *Load Detection*

To accurately detect the load status of a broker, the load metric and related thresholds need to be clarified. In the H-E broker model, brokers are classified into a cluster-head broker and cluster-edge broker, and different load metrics and thresholds are allocated to the different types of brokers.

1) *Load Metrics for Head Broker and Edge Broker*

The main tasks of a B_h are: to route messages from publishers and B_h of other clusters to the B_e that serves matched subscribers; to route messages from publishers to the B_h of another clusters that serve matched subscribers. As claimed in [1], a B_h is less likely to be overloaded for doing the matching work as no subscribers connect to it. Therefore, the load status of a B_h is mainly affected by the network bandwidth usage. Table I lists the load metrics used for cluster-head broker.

TABLE I. LOAD METRICS FOR THE HEAD BROKER

Metric	Expression
Downlink Utilization	Input-Rate / Downlink-Bandwidth
Uplink Utilization	Output-Rate / Uplink-Bandwidth

B_e serves all subscribers, and therefore does a lot more matching work. So, the load matching costs need to be monitored. In addition, since a guaranteed end-to-end transmission delay is required, a Queue Depth metric that measures the number of messages waiting in the output queue and reflects the message waiting time in a broker is introduced. Table II lists the load metrics used for cluster edge broker.

TABLE II. LOAD METRICS FOR EDGE BROKER

Metric	Expression
Downlink Utilization	Input-Rate / Downlink-Bandwidth
Matching Utilization	Input-Rate / Matching-Rate
Uplink Utilization	Output-Rate / Uplink-Bandwidth
Queue Depth	No. of Messages waiting in each Output Queue

2) *Threshold Determination*

We introduce two thresholds for each metric to describe the load status of a broker. A lower threshold (TH_{low}) indicates whether or not a broker is available to accept more loads, while a higher threshold (TH_{high}) indicates whether or not load shifting is required. Based on the two thresholds, the load status of a broker is divided into LOW LOAD, HIGH LOAD, and OVERLOAD. The relationship between the threshold and the load status is defined in Table III.

TABLE III. LOAD STATE & THRESHOLD

Condition	Status
(All the metrics) $< TH_{low}$	LOW LOAD
$TH_{low} <$ *(Any metric) & (All the metrics)* $< TH_{high}$	HIGH LOAD
$TH_{high} <$ *(Any metric)*	OVERLOAD

The higher value the HIGH LOAD threshold is set to (e.g., 99% CPU Utilization), the more the system resources can be used. However, a broker can become overloaded before it can do any offloading. The magnitude of the difference between the lower and higher threshold controls the efficiency of load balancing and the level of the load imbalance between brokers. For example, a small difference, e.g., 1%, reduces the load imbalance between brokers but makes brokers more likely to enter OVERLOAD from HIGH LOAD, which may result in endless load balancing cycles [1]. In addition, based on whether or not the load metrics are affected by the delay sensitivity of the messages, the load metrics are divided into two groups and assigned with different thresholds.

Both uplink usage and downlink usage for B_h are set with static thresholds, i.e., $TH_{low} = 0.9$ and $TH_{high} = 0.95$. The same thresholds are applied to the downlink utilization and the matching utilization for the edge broker. These values are retrieved from the threshold defined for PEER [1]. The uplink usage and the Queue Depth metric of a B_e are considered separately as they affect the time of messages waiting in the broker. In this design, only TH_{low} is assigned to the uplink utilization metric of the edge broker as it is only used to indicate whether or not the broker is available for more loads, and only TH_{high} is set for the Queue Depth metric that is used to trigger load balancing with latency guarantees. The value of TH_{low} for the uplink utilization of the edge broker is set the same as others, e.g., 0.9, while the value of TH_{high} for Queue Depth of edge broker is calculated based on the end-to-end latency requirements for different topics of individual subscribers, the transmission delays, and the time a message spent in brokers. The following procedure shows the steps of determining the dynamic TH_{high} for Queue Depth metric.

a) *Transmission Time*

The end-to-end latency requirement for subscriber "s" on topic "T" is denoted as $t_{s,T}$. The practical end-to-end latency is calculated as the sum of the total transmission time (t_{s,T_trans}) and the total time spent in broker (t_{s,T_broker}). With the H-E model, the total transmission time is obtained based on the transmission time from publishers to B_h ($t_{s,T\_p\text{-}h}$), from B_h to B_h of matching clusters ($t_{s,T\_h\text{-}h}$), from B_h to B_e of the matched subscribers ($t_{s,T\_h\text{-}e}$), and from B_e to subscribers ($t_{s,T\_h\text{-}s}$), i.e., $t_{s,T\_trans} = t_{s,T\_p\text{-}h} + t_{s,T\_h\text{-}h} + t_{s,T\_h\text{-}e} + t_{s,T\_e\text{-}s}$. For the case that publisher clients on the same topic are served by different clusters, the transmission time obtained for different publishers may have different values since the time cost from publishers to B_h and from B_h to B_h may be different. In our design, the maximum transmission time from all the obtained transmission time is selected, denoted as $t_{s,T\_trans\text{-}sel}$.

b) Time in Broker

The total time spent in broker (t_{s,T_broker}) consists of the time spent in B_h that serves the publisher (t_{s,T_h}), the time spent in the remote B_h belonging to the matched clusters ($t_{s,T\_remote\text{-}h}$), the B_e that serves the matched subscribers ($t_{s,T\text{-}e}$), i.e., $t_{s,T\_broker} = t_{s,T\_h} + t_{s,T\_remote\text{-}h} + t_{s,T\_e}$. For each broker, the time cost is the sum of the arrival time ($t_{s,T\_arrival}$), departure time ($t_{s,T\_departure}$), the matching time ($t_{s,T\_matching}$) and the time waiting in the queue ($t_{s,T\_waiting}$). Each of the arrival and departure time is determined by the size of the message and the uplink/downlink bandwidth, and the matching time is mainly affected by the number of filters in the matching process. The waiting time in a broker is determined by the number of messages waiting in the queue and the message output rate.

c) Dynamic Threshold

With the end-to-end transmission delay, the maximum time that a message can spend in the output queue of broker B_e ($t_{s,T\text{-}e}$) can be determined as $t_{s,T} - t_{s,T\_trans\text{-}sel} - t_{s,T\_h} - t_{s,T\_remote\text{-}h} - t_{s,T\_arrival\text{-}e} - t_{s,T\_matching\text{-}e} - t_{s,T\_departure\text{-}e}$. This maximum-allowed time a message can spend in the output queue varies due to the change of transmission time, matching time and arrival/departure time. This maximum waiting time in the message broker is used to compute the higher threshold for Queue Depth metric for subscriber "s" on topic "T", i.e., the value of Queue Depth at a time t_i ($QD_{s,T}(t_i)$) must follow the condition defined in (1), where $\lambda_{s,T}(t_{i+1})$ and $\mu_{s,T}(t_{i+1})$ are the predicted message input rate and output rate in message/s for time t_{i+1}, and t_{LB} is the average time cost for load balancing that is mainly affected by the notification message transmission time from HOM to subscribers, e.g., from milliseconds to seconds, and the analysis time, e.g., in milliseconds.

$$\frac{QD_{s,T}(t_i)+[\lambda_{s,T}(t_{i+1})-\mu_{s,T}(t_{i+1})]}{\mu_{s,T}(t_{i+1})} \le t_{s,T-e} - t_{LB} \quad (1)$$

Therefore, the higher threshold for Queue Depth at time t_i ($TH_{s,T}(t_i)$) is found with (2).

$$\begin{aligned} TH_{s,T}(t_i) = \left(t_{s,T-e} - t_{LB}\right) * \mu_{s,T}(t_{i+1}) \\ - [\lambda_{s,T}(t_{i+1}) - \mu_{s,T}(t_{i+1})] \end{aligned} \quad (2)$$

C. Load Analysis

Load analysis is invoked when a broker is overloaded. A load analyser aims to estimate and profile the load distribution for individual clients served by the broker, and prioritizes the offloading clients according to their overloaded load metrics.

1) Load Estimation

Both ELA and HLA compute the network bandwidth usage for individual clients based on the message exchange rate and the bandwidth, e.g., the uplink usage of edge broker for subscriber "s" on topic "T" is computed as the message output rate ($\mu_{s,T}$) / uplink bandwidth. In addition, ELA estimates the matching utilization and records the Queue Depth for each subscriber on each topic.

2) Priorities Offloading Client

In our design, the clients of the same topic are recognized as a bundle in the offloading process, i.e., they are either migrated together to the load-accepting broker or kept together in the overloaded broker. Only if the load-accepting broker cannot accept any bundle of clients, these clients are dealt with separately.

In the head broker, the publishers of different topics can be categorized into four groups: the publishers that only have remote subscribers (P_r), the publishers that have both local and remote subscribers ($P_{r\text{-}l}$), the publishers that only have local subscribers (P_l), and the publishers that have no subscribers (P_n). So, if the broker is in a downlink overload state, the priority of all the publishers are $P_n > P_r > P_{r\text{-}l} > P_l$, while if it is an uplink overload state, the priority relationship becomes $P_r > P_{r\text{-}l} > P_l > P_n$. The difference between the two is the location of P_n, because migrating publishers with no subscribers cannot reduce the uplink utilization but only reduce the downlink utilization.

In each edge broker, similar to the equivalent situation with head brokers, the subscribers on different topics can be categorized into S_r, $S_{r\text{-}l}$, S_l and S_n. In addition, for the Queue Depth metric, as it does not relate to the locations of the publishers, the subscribers are categorized into three groups: subscribers without message waiting in the queue (S_{empty}), subscribers with message waiting in the queue but not overloaded ($S_{w\text{-}no}$), and subscribers of which the Queue Depth metric is overloaded ($S_{overload}$). In all the groups above, the subscribers are ordered based on its allowed waiting time, i.e., the larger the waiting time, the higher the priority. The relationship between subscribers is defined in Table IV.

TABLE IV. PRORITIES SUBSCRIBERS IN EDGE BROKER

Overload Metric	Priority
Downlink Utilization	$S_r > S_{r\text{-}l} > S_l > S_n$.
Uplink Utilization	
Matching Utilization	$S_n > S_r = S_{r\text{-}l} = S_l$
Queue Depth	$S_{empty} > S_{w\text{-}no} > S_{overload}$

D. Load Balancing

After the load analysis process, load balancing takes place. As described in PEER, if a head broker becomes overloaded, load balancing happens between head brokers in different clusters by migrating publishers from an overloaded head broker to head brokers with lesser loads. If instead, the edge broker becomes overloaded, the load balancing first takes place within a local cluster. Only if there is no available load-accepting broker in the local cluster, i.e., no broker is in the LOW LOAD state, or the available load-accepting brokers have less load capacity than that required by the overloaded broker to recover from OVERLOAD state, is inter-domain load balancing invoked. All the load balancing processes follow a similar three-step offloading strategy, i.e., Load-Accepting Broker Locating, Client Selection, and Client Migration. In this paper, intra-domain load balancing between edge brokers is described below as an example.

1) Load-Accepting Broker Locating

The EOM of an overloaded broker checks the load state of brokers in the same domain to locate brokers in a LOW LOAD state and sends a load balancing request to a HOM with the candidate broker ID(s). The HOM records whenever a broker is in a load-balancing phase and sends requests to all candidate brokers. The EOMs of these candidate brokers report the values of all the load metrics to the HOM. And when HOM receives this information, it will in turn forward to the requesting EOM.

2) Client Selection

Based on the results of step 1, EOM of the overloaded broker prioritizes the candidate brokers based on the value of the overloaded load metric of the broker, i.e., the broker with the lowest value of the load metric has the highest priority to accept the load. In addition, from the prioritized client list, EOM retrieves the clients and estimates the load influence to the load-accepting broker for all the load metrics, e.g., for the uplink bandwidth usage, the influence is estimated as the (input rate of the client / the uplink bandwidth of the load-accepting broker), which means that if the clients are migrated to the load-accepting broker, the uplink usage will be increased by this amount. So, in the case that the client does not overload the load-accepting broker, it is selected and put in an offloading list. The selection process continues until the estimated load status of the overload broker is not OVERLOAD any more. The offloading list is then sent to the HOM. HOM notifies the EOMs of the selected edge brokers to be in a load-balancing phase.

3) Client Migration

In the last step, HOM sends messages to all the clients that are in the offloading list, asking them to start a message exchange via the load-accepting broker(s). All the clients then set up connection(s) to the load-accepting broker(s) and drop the connection to the offloading broker except for subscribers that have messages waiting in the queue. In this case, the subscribers will drop the connections only when all the messages waiting in the overloaded broker are received. In addition, a message is sent by each client to HOM to confirm the completion of the migration process. HOM counts the number of clients that have completed the migration away from the overloaded broker. There is also a default timeout for the migration so that the load-balancing phase can stop even if some clients stop the message exchange during the migration. When all the clients complete the migration or the waiting time has timed out, the HOM notifies all the EOMs involved in the load-balancing phase that the load balancing is complete.

V. VALIDATION

We validate our framework by comparing our load balancing mechanism to that designed for the PEER framework. In this paper, a local load balancing triggered by Queue Depth metric is given as an example. The setup used for the local load balancing experiment involves four edge brokers (B_0, B_1, B_2, and B_3) connected to one cluster-head broker (B_h) to form a star topology, which forms a messaging bus to exchange information in an EWS. The simulation environment specification is listed in Table V. For each broker, the uplink bandwidth and downlink bandwidth is the same and is static during the experiment so that the broker-to-broker transmission latency will not change, e.g., is set at 0.1s. In addition, we assume that the client to broker transmission latency is also constant during the experiment, e.g., 0.2s.

TABLE V. SIMULATION EXPERIMENT SPECIFICATION

Broker ID	**Specifications**		
	CPU (MHz)	***Memory (MB)***	***Bandwidth (Mbps)***
B_h	2000	64	20
B_0	800	32	6.5
B_1	1500	32	8
B_2	1300	64	5
B_3	1000	64	8

Messages for 15 topics are published, i.e., in the EWS system, 15 types of data are exchanged through the messaging bus. The number of publishers for each topic is a random number, e.g., 1-5. Each publisher publishes messages in an average rate of 50 message/s. The number of subscribers for each topic is a random number, e.g., 1-8. In the experiment, we assume that subscribers of different topics have different end-to-end latency requirements but the subscribers of the same topic have the same requirement. The average message size changes for different topics, e.g., from 200 Byte to 1KB. Table VI gives an example of how topics are specified in one experiment.

TABLE VI. TOPIC SPECIFICATIONS IN ONE EXPERIMENT

Topic ID	**No. of Pubs**	**No. of Subs**	**Latency Requirement (s)**	**Msg Size (Byte)**
1	1	8	1.8	200
2	2	2	1.7	800
3	5	1	1.6	1000
4	4	2	1.5	400
5	3	3	1.4	200
6	1	1	1.3	400
7	2	5	1.2	300
8	2	7	1.1	400
9	5	2	1.0	500
10	4	4	0.9	200
11	1	5	30	600
12	3	2	60	400
13	1	6	40	200
14	2	3	50	300
15	4	5	100	200

The reason to use a random number is to allow the broker loads to be varied in different experiments to improve the validation. On the other hand, the reason to have such a range, e.g., 1-5 for publisher, is to lower the chance that all the brokers become overloaded since in that case load balancing is not useful - more brokers are required.

According to the end-to-end transmission latency requirements and the assumptions for the static client-to-broker and broker-to-broker transmission delay, the maximum time of a message can be held in a broker can be determined, e.g., for topic 1, $t_{topic1\text{-}broker} = 1.8 - 0.2 - 0.1 = 1.5s$. These values are used in experiment to determine higher threshold for the Queue Depth metric.

In experiment start-up, all brokers are instantiated simultaneously with the MAs. After that, all publishers register and connect to head brokers, and MAs start to measure the load status of a broker and the broker-to-broker transmission delays. Each experiment is divided into three phases: 1) client distribution phase: 1s – 15s, subscribers of each topic in EWS are registered and distributed to the available brokers in each second; 2) equilibrium phase: 15s - 29s, both publishers and subscribers in EWS are running without message bursts and client joining or leaving; 3) message burst simulation and load balancing phase: at 30s, a burst that simulates a message flood when a crisis detected is generated by doubling the speed of publishing 7 topics (e.g., topic 2, 4, 5,..., 12, 14); after 31s, up to the end of the experiment, load balancing will be triggered if any load metric exceeds its higher threshold. The reason to set time slots to these values is to highlight the changes in each stage of the simulation. The experiment can be easily expanded by 1) adding more brokers, publishers and subscribers; 2) increasing the time intervals for each phase; 3) generating more message bursts.

Figure 3 shows the simulation results for the uplink utilization in percent (y) against time in second (x). The value above 100% indicates that the output queue starts to build up.

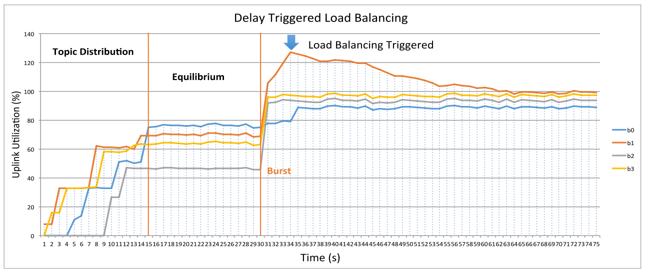

Figure 3. Simulation Result for Uplink Utilization

After the workload distribution, broker b1 serves topics 1, 3, 8, and 14 (refer to the 4 inflection points of b1 in the topic distribution stage). In addition, for b1, the output queue starts to build up after a message burst (30s) as the uplink utilization exceeds 100%; 4s after this (34s), the queue depth value of topic 8 exceeds the TH_{high}, and thus load balancing is triggered. Topic 1 in b1 is migrated to broker b0. Therefore, broker b1 has more bandwidth to clear the messages for topic 8 in the queue (from 34s – 62s, a balancing stage). After 62s, the message queue for topic 8 in broker b1 is removed. The uplink utilizations for all the brokers are below 100%. Figure 4 shows the Queue Depth, i.e., number of messages in the output queue, for topic 8 in broker b1.

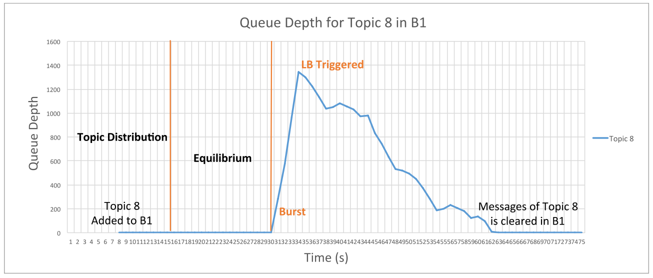

Figure 4. Queue Depth for Topic 8 in broker b1

When the same simulation is applied using PEER load balancing mechanism, the results are shown in Figure 5.

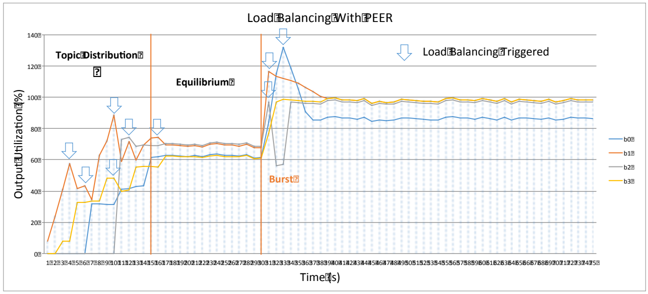

Figure 5. Simulation Result for Uplink Utilization for PEER

In the topic distribution phase (1s-15s) of PEER, all the subscribers are initially connected to broker b1 and migrated to other brokers (e.g., b2) based on the load differences (as there is no work distribution mechanism in their work). In addition, after a burst (30s), as the delay requirements for topics are ignored, unnecessary load balancing takes place between broker b0 and b2 (at time 31s), that results in an additional load balancing to balance the two at time 33s. Comparing Figure 3 to Figure 5, the differences indicate that our proposed delay-aware load balancing method is more effective in workload distribution, and can avoid unnecessary load balancing as the delay requirements are considered.

VI. CONCLUSION AND FUTURE WORK

In this paper, an analysis of existing load management solutions for PSMOM was presented. Existing solutions ignored the end-to-end delay requirements, which may introduce unexpected delays for delay sensitive messages or trigger unnecessary load balancing that introduces extra overhead to the system, and therefore they were not applicable for PSMOM in EWS. To address the above limitations, we proposed a delay sensitive load management solution that extends an existing state-of-the-art, PEER framework [1]. In addition, an intra-cluster load balancing example was presented with comparison to PEER and the results showed that the proposed framework is aware of the delay requirements, and has the potential to efficiently solve the broker overload problem in a LAN-based setting.

The framework was implemented with Apache Qpid [14], an open source AMQP based MOM product. In the future, real sensor data from the TRIDEC project will be adopted to evaluate the framework in a WAN-based setting.

ACKNOWLEDGMENT

This work is supported in part by the EU FP7 funded project TRIDEC (FP7-258723-TRIDEC) and by a PhD studentship at Queen Mary University of London.

REFERENCES

[1] A. K. Y. Cheung and H.-A. Jacobsen, "Load Balancing Content-based Publish/Subscribe Systems", ACM Transactions on Computer Systems, Vol. 28, Issue 4, Article 9, 55 pages, Dec. 2010, doi: 10.1145/1880018.1880020.

[2] I. Aekaterinidis and P. Triantafillou, "PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network", 26th IEEE International Conference on Distributed Computing Systems (ICDCS 06), Jul. 2006, pp. 23-32, doi:10.1109/ICDCS.2006.63.

[3] P. Tran and P. Greenfield, "Behavior and Performance of Message-Oriented Middleware Systems", Proc. 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW 02), Jul. 2002, pp. 645-650, doi:10.1109/ICDCSW.2002.1030842.

[4] A. K. Y. Cheung and H.-A. Jacobsen, "Dynamic Load Balancing in Distributed Content-based Publish/Subscribe", Proc. 7th ACM/IFIP/USENIX International Conference on Middleware (Middleware 06), Nov. 2006, pp. 141-161.

[5] J. Wang, J. Bigham, and J. Wu, "Enhance Resilience and QoS Awareness in Message Oriented Middleware for Mission Critical Applications", 8th International Conference on Information Technology: New Generations (ITNG 11), Apr. 2011, pp. 677-682, doi: 10.1109/ITNG.2011.120.

[6] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: Content-based Publish/Subscribe over P2P Network", Proc. 5th ACM/IFIP/USEaNIX International Conference on Middleware (Middleware 04), Oct. 2004, pp. 254-273.

[7] Y. Chen and K. Schwan, "Opportunistic Overlays: Efficient Content Delivery in Mobile Ad Hoc Networks", Proc. 6th ACM/IFIP/USENIX International Conference on Middleware (Middleware 05), Nov. 2005, pp. 354-374, doi: 10.1007/11587552_18.

[8] E. Casalicchio and F. Morabito, "Distributed Subscriptions Clustering with Limited Knowledge Sharing for Content-Based Publish/Subscribe Systems", 6th IEEE International Symposium on Network Computing and Applications (NCA 07), Jul. 2007, pp. 105-112, doi: 10.1109/NCA.2007.16.

[9] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and L. Zhang, "Clustering Algorithms for Content-Based Publication-Subscription Systems", Proc. 22nd International Conference on Distributed Computing Systems (ICDCS 02), May 2002, pp. 133-142, doi:10.1109/ICDCS.2002.1022250.

[10] A. Riabov, Z. Liu, J. L. Wolf, P. S. Yu, and Li Zhang, "New Algorithms for Content-Based Publication-Subscription Systems", Proc. 23rd International Conference on Distributed Computing Systems (ICDCS 03), May 2003, pp. 678-686, doi: 10.1109/ICDCS.2003.1203519.

[11] T. Wong, R. H. Katz, and S. McCanne, "An Evaluation of Preference Clustering in Large-scale Multicast Applications", 9th IEEE International Conference on Computer Communications (INFOCOM 00), Mar. 2000, pp. 451-460, doi:10.1109/INFCOM.2000.832218.

[12] Y. Zhu, "Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services", IEEE Transactions on Parallel and Distributed Systems, Vol. 18, May 2007, pp. 672-685, doi:10.1109/TPDS.2007.1012.

[13] F. Paganelli, G. Vannuccini, D. Parlanti, D. Giuli, and P. Cianchi, "GEMOM Middleware Self-healing and Fault-tolerance: a Highway Tolling Case Study," The Sixth International Conference on Systems and Networks Communications (ICSNC 11), Oct. 2011, pp. 136-142.

[14] Apache Qpid, Official Web Page, http://qpid.apache.org (last access date: September 24th, 2013)