Green Resource Allocation Algorithms for Publish/Subscribe Systems

Alex King Yeung Cheung and Hans-Arno Jacobsen
University of Toronto, Middleware Systems Research Group (MSRG.org)
Email: {cheung,jacobsen}@eecg.utoronto.ca

***Abstract*—A popular trend in large enterprises today is the adoption of green IT strategies that use resources as efficiently as possible to reduce IT operational costs. With the publish/subscribe middleware playing a vital role in seamlessly integrating applications at large enterprises including Google and Yahoo, our goal is to search for resource allocation algorithms that enable publish/subscribe systems to use system resources as efficiently as possible. To meet this goal, we develop methodologies that minimize system-wide message rates, broker load, hop count, and the number of allocated brokers, while maximizing the resource utilization of allocated brokers to achieve maximum efficiency. Our contributions consist of developing a bit vector supported resource allocation framework, designing and comparing four different classes with a total of ten variations of subscription allocation algorithms, and developing a recursive overlay construction algorithm. A compelling feature of our work is that it works under any arbitrary workload distribution and is independent of the publish/subscribe language, which makes it easily applicable to any topic and content-based publish/subscribe system. Experiments on a cluster testbed and a high performance computing platform show that our approach reduces the average broker message rate by up to 92% and the number of allocated brokers by up to 91%.**

I. Introduction

Publish/subscribe is commonly used in enterprises as a messaging substrate for event dissemination. Real-world examples include GooPS [1], Google's publish/subscribe system that integrates its web applications; SuperMontage [2], Tibco's publish/subscribe distribution network for NASDAQ's quote and order-processing system; and GDSN (Global Data Synchronization Network) [3], a global publish/subscribe network that allows suppliers and retailers to exchange timely and accurate supply chain data. At the same time, 97% of today's enterprises are actively engaged in green computing practices to (1) reduce IT maintenance costs, (2) reduce the carbon footprint, and (3) promote an environmentally responsible brand image of the company [4]. Seeing how publish/subscribe is so intricately tied to enterprises that also have strong green IT initiatives, this paper presents resource allocation algorithms that allocates as few brokers as possible for any given workload, while maximizing the resource utilization of allocated brokers.

In order to minimize the number of brokers, we need to minimize the amount of messages forwarded and processed in the system. Reducing the broker count also reduces the network size, which in turn improves publication hop count. To satisfy all these requirements, we develop a 3-phase scheme to reconfigure the publish/subscribe system. In Phase 1, we gather performance and workload information from the network using bit vectors. In Phase 2, we allocate the subscriptions to brokers using the information gathered from Phase 1. In Phase 3, we recursively construct the broker overlay with the subscriptions already allocated. Finally, we strategically place the publishers onto the newly built broker overlay with a publisher relocation algorithm called GRAPE [5]. The entire process shows that we are altering three variables to meet our optimization criteria, which is proven to be an NP-complete problem [6]. In our evaluation, we compare our algorithms against three related and two baseline approaches which are representative of typical publish/subscribe deployments where the measure of a ”good” topology is not easily quantifiable [7], [8], [9], [10].

To the best of our knowledge, we are among the first to minimize the number of brokers in content-based publish/subscribe systems. There has been prior work that focused on improving system performance (i.e., reducing the in-network processing and/or hop count) by relocating subscriptions [11], publication sources [5], and brokers [7], [12], [13]. However, these prior approaches only manipulated one variable, either brokers, publishers, or subscribers to achieve their optimization criteria. In contrast, our work manipulates all three variables to achieve the same objective while minimizing the total number of allocated brokers in the system.

The resource allocation algorithms that we develop are primarily targeted at enterprise-grade messaging systems consisting of hundreds of dedicated servers within a data center environment [1], [14], [15]. These systems include the commercial publish/subscribe systems mentioned previously as well as other systems enabled by publish/subscribe such as workflow management systems [16], business activity monitoring [17], network/systems monitoring [18], decentralized business process execution [19], RSS dissemination [20], and resource discovery [21].

The main contributions of this paper are: (1) a bit vector supported resource allocation framework (Section III), (2) three subscription allocation algorithms that account for broker capacities with one of them capable of clustering subscriptions of similar interests (Section IV), (3) a recursive broker overlay construction algorithm (Section V), and (4) real world evaluation of the baseline, related, and proposed approaches

Accepted in the 31th IEEE International Conference on Distributed Computing Systems (ICDCS), 2011

implemented on the PADRES [22] open source content-based publish/subscribe system (Section VI).

II. Related Work

Related work to our contributions can be classified into three areas: (1) publish/subscribe systems, (2) broker and client reconfiguration algorithms, and (3) subscription clustering.

A. Publish/Subscribe Systems

Two main classes of distributed content-based publish/subscribe systems exist today: filter-based [8], [9], [10], [16], [19] and multicast-based [23], [24], [25]. In the filter-based approach, advertisements and subscriptions are propagated into the network to establish paths that guide publications to subscribers. Each publication is matched at every broker along the overlay to get forwarded towards neighbors with matching subscriptions. Consequently, the farther the publication travels, the higher is the delivery delay. In the multicast-based approach, subscribers with similar interests are clustered into the same multicast group. Each publication is matched once to determine the matching multicast group(s) to which the message should be multicasted, broadcasted, or unicasted. As a result, matching and transmission of a publication message happen at most once, thus incurring minimal delivery delay. However, compared to the filter-based approach, subscribers may receive unwanted publications because subscriptions with even slightly different interests may still be assigned to the same multicast group.

In a sense, our approach is trying to build multicast groups within a filter-based publish/subscribe system by clustering subscribers of similar interests together with the matching publishers on a broker overlay tailored to the clusters. At the same time, we guarantee no false-positive publication delivery and do not have to manage and partition subscribers into multicast groups. We can also adapt our solution to systems where clients take on both publisher and subscriber roles by separating the network connections between the two entities.

B. Broker Overlay and Client Placement Reconfigurations

A number of approaches in the literature also try to reduce the distance between publishers and subscribers on an overlay with dedicated brokers. Baldoni *et al.* [7], Jaeger *et al.* [12], and Migliavacca *et al.* [13] dynamically reconfigure inter-broker overlay links to allow publications to skip over brokers with no matching subscribers, called *pure forwarders*, and shorten the publication delivery path. Cheung *et al.* [5] adaptively place publishers in the broker overlay to position them closer to subscribers. However, these approaches cannot reduce the overall system message rate if at least one subscriber subscribes to the same subscription at every broker. Our solution addresses this limitation by not only reconfiguring the broker overlay, but also relocating both publishers and subscribers. Our experiment results indeed show that under such scenario, relocating only publishers have no impact on the broker system message rate, while our approach achieves reductions of up to 92%.

C. Subscription Clustering

Riabov *et al.* [24] utilize the concept of clustering to group similar subscriptions together into multicast groups to eliminate pure forwarders. SUB-2-SUB [26] and Rappel [27] cluster subscribers of similar interests and propagate publications only among interested peers. However, Riabov *et al.*, SUB-2-SUB, Rappel, and Gryphon's [10] cluster algorithms differ from our work in three significant ways. First, their clustering algorithms do not take resource constraints into consideration. For instance, in SUB-2-SUB and Rappel, there is no hard bound on the number of clients in a peer-to-peer group. In [24], there is no restriction on the number of clients assigned to a multicast IP address. In Gryphon, all subscriptions are stored in main memory anyway. Second, their approaches assume range or point subscription queries, whereas our approach is completely language independent. This allows our solution to be readily applicable to not only range query subscriptions, but also to queries with negation, string operators, XPath expressions [28], or graph expressions [20]. Third, the peer-to-peer architectures of SUB-2-SUB and Rappel are fundamentally different from the dedicated broker architecture of our work. Publish/subscribe clients in SUB-2-SUB and Rappel are capable routing of messages because they assume the role of brokers as well. In our work, clients do not assume the role of brokers; therefore, only brokers are capable of routing messages. Fourth, the cluster algorithms employed in [24], such as *pairwise*, requires one to specify the number of clusters a priori. Our approach, on the other hand, computes the number of clusters at runtime based on the subscriptions' interests and resource constraints of brokers. In our evaluation, we compare our approach with two extended versions of the pairwise algorithm from [24] and the clustering measurement metric from [10].

III. Phase 1: Resource Allocation Framework

Our strategy to minimize the number of active brokers consists of three phases. Phase 1 gathers performance and workload information from the brokers in order to carry out computations in Phases 2 and 3. Phase 2 assigns subscriptions to brokers using a subscription allocation algorithm. Phase 3 recursively constructs the broker overlay using the subscription allocation strategy from Phase 2. After Phase 3, GRAPE relocates the publishers from the center of the network to where the matching subscribers reside.

The components required to support Phases 1 to 3 are:

- An external publish/subscribe client called *Coordinator for Reconfiguring the Overlay and Clients* (CROC) that connects to any broker in the overlay to collect information about the currently deployed system, executes Phases 2 and 3, and orchestrates the reconfiguration.
- Integrated into each broker is the CROC Back-end Component (CBC) that responds to commands sent by CROC, such as responding to information requests, profiling of subscribers, etc.

A. Information Gathering

The information gathering protocol can be implemente an out-of-band messaging protocol or using publish/subsc that is already supported by the system. We chose the l methodology to avoid the additional complexity of a messaging layer. When CROC connects to a broker on overlay, it sends a *Broker Information Request* (BIR) mess to the first broker. Whenever a broker receives a BIR mess it broadcasts the BIR message to all of its neighbors. Bro reply to the BIR message with a *Broker Information Ans* (BIA) message only if it has no neighbors to forward the message or have received the BIA messages of all neighl to which it forwarded the BIR message. The latter condi enables the aggregation of received BIA messages with current broker's into one BIA message to reduce overh The BIA message contains the following information al the broker:

- URL - This is needed for reassigning subscribers broker neighbors in Phases 2 and 3
- Matching delay function - A linear function that models the matching delay as a function of the number of subscriptions. This enables CROC to predict the input load of the broker during subscription assignment and overlay construction in Phases 2 and 3
- Total output bandwidth - CROC uses this to predict the output load of the broker during subscription assignment and overlay construction in Phases 2 and 3
- Set of local subscriptions and their profiles - CROC will relocate these subscriptions in Phase 2 based on their information profile
- Set of local publishers and their profiles - CROC uses this information to predict the load imposed by each subscription

Once CROC obtains all BIA messages from every broker in the system, it executes Phases 2 and 3 to reassign subscriptions to brokers and reconfigures the broker overlay. The results of the reassignment is in the form of publications directed to each broker controlling where publishers and subscribers should migrate, and which neighbors brokers should connect with.

B. Subscription and Publisher Profiles

A subscription profile captures the publications sinked by a subscription in the recent past. This allows CROC to accurately estimate the load requirements of subscriptions without any assumptions on the workload distribution (i.e., Gaussian, Zipf, etc.) and cluster subscriptions independent of the publish/subscribe language. Profiles for subscriptions are generated and maintained by the CBC at the subscribers' immediate brokers.

A subscription profile consists of one or more bit vectors each associated with a counter variable. The bit vector is a medium to record the publications that this subscription received from a specific publisher using as little memory space as possible. Thus, there is one bit vector for every unique publisher. Using the left side of Figure 1 as an example, subscription S_1 has two bit vectors because it received publications from two publishers, *Adv1* and *Adv2*. Each publisher appends a message ID, which is just an integer counter, as well as its globally unique advertisement ID into its publication messages, which serves to identify the publisher of every publication. An integer counter is associated with each bit vector to indicate the ID of the first bit in the bit vector. Naturally, a set bit in the bit vector corresponds to the subscription having received that particular publication. Using Figure 1 as an example, subscription S_1 received publications with IDs 75, 76, and 77 from publisher *Adv1* and publications with IDs 144 to 148 from publisher *Adv2*.

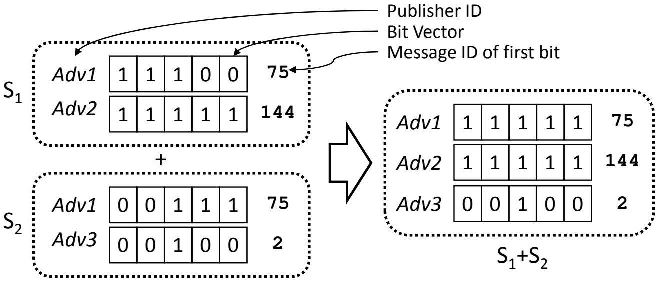


Fig. 1. Clustering two subscriptions to form a new subscription

Bit vectors have bounded size, whose default value is 1,280. A larger size will improve the accuracy of estimating the anticipated load of a subscription, but will lengthen the time required to profile subscriptions (i.e., fill up the bit vector). If on receiving a publication, the bit to set in the bit vector exceeds the size of the bit vector length, then the bit vector is shifted just enough to record the publication in the last bit of the bit vector while updating the integer variable by the number of bits shifted. Depending on the implementation, if the first bit starts at the most significant bit in memory, then one would shift the bit vector to the left, or vice versa. For example, if the bit vector length is 10 while the counter representing the first bit is 100, and an incoming publication has a publication ID of 119, then shift the bit vector by 10 bits, set the bit at index 9, and update the counter to 110.

A publisher profile contains the publisher's advertisement ID, publication rate, bandwidth consumption, and the message ID of the last publication message sent. The second and third pieces of information are used for estimating the load requirements of subscriptions. The last piece of data is used for synchronizing the message ID counter in all bit vectors that correspond to the same publisher. As an example, to estimate the bandwidth requirement of a subscription with 10 out of 100 bits set in a bit vector corresponding to a publisher whose publication rate is 50 msg/s and bandwidth is 50 kB/s, the publication rate induced by this subscription is 5 msg/s and the output bandwidth requirement of this subscription on a broker is 5 kB/s.

IV. Phase 2: Subscription Allocation Algorithms

We developed three subscription allocation algorithms of different complexities to allocate subscriptions to a minimal set of brokers: Fastest Broker First (FBF), BIN PACKING, and Clustering with Resource Awareness and Minimization (CRAM). We will refer to the former two approaches as *sorting* *algorithms*. The set of subscriptions to allocate consists of all the subscriptions reported in the BIA messages in Phase 1. We call this set of subscriptions the *subscription pool*. The set of brokers on which to allocate the subscriptions include all the brokers that sent a BIA message back to CROC. We call this set of brokers the *broker pool*. The outcome at the end of Phase 2 are a set of non-connected brokers where some have subscriptions allocated to them and some do not.

A. FBF

In FBF, brokers in the broker pool are first sorted in descending resource capacity. From our experiences in working with an open source publish/subscribe system [22] on cluster and Internet scale testbeds, the bottleneck of a broker is not the processing but the forwarding of messages, that is, the network I/O. Thus, we first sort the brokers in the broker pool in descending order of total available output bandwidth. Next, a subscription is randomly removed from the subscription pool and is assigned to the next most resourceful broker that has the capacity to handle the subscription. A broker is deemed to have enough capacity to handle a subscription only if by accepting this subscription, its remaining available output bandwidth is greater than 0 and its incoming publication rate is less than or equal to its maximum matching rate. The maximum matching rate is calculated by taking the inverse of the matching delay computed using the matching delay function supplied in the BIA message. The algorithm ends when all subscriptions from the subscription pool are allocated to the brokers or if at least one subscription cannot be allocated to any broker. Assuming that the number of subscriptions is much larger than the number of brokers, the complexity of this algorithm is O(S) where S is the total number of subscriptions in the system.

B. BIN PACKING

BIN PACKING is similar to FBF except that instead of randomly picking subscriptions and assigning them to brokers, subscriptions in the subscription pool are first sorted in descending order of bandwidth requirement. Then, the algorithm repeatedly allocates the next subscription with the highest bandwidth requirement to the next most resourceful broker that has the capacity to handle the subscription. The algorithm ends when all subscriptions from the subscription pool are allocated to the brokers or if at least one subscription cannot be allocated to any broker. Due to the sorting of the subscriptions, the complexity of this algorithm is O(Slog(S)). Our experiments show that BIN PACKING consistently allocates one less broker than FBF, which is inline with theoretical expectations [29].

C. CRAM

CRAM is significantly different from the two prior sorting algorithms because it clusters the next closest pair of subscriptions before allocating them to brokers. The closeness between two subscriptions can be measured by the following four possible closeness metrics, where S_1 and S_2 represent the bit vector profiles of two arbitrary subscriptions.

- INTERSECT: $|S_1 \cap S_2|$ - cardinality of the intersection
- XOR: $|S_1 \oplus S_2|^{-1}$ - inverse of the xor'ed cardinality with a capped maximum value to handle division by zero. This is derived from the closeness metric in Gryphon [10] to make it consistent with the other metrics (i.e., higher magnitude is more favorable).
- IOS: $\frac{|S_1 \cap S_2|^2}{|S_1|+|S_2|}$ - cardinality of the intersection squared over the sum of the cardinalities
- IOU: $\frac{|S_1 \cap S_2|^2}{|S_1 \cup S_2|}$ - cardinality of the intersection squared over the cardinality of the union

Ideally, the two subscriptions that share the highest overlap of publication traffic while introducing the least amount of non-overlapping traffic is desired. The INTERSECT metric is the simplest and it accounts for the former but misses the latter property. The XOR metric accounts for the latter but misses the former property. In addition, the XOR metric cannot identify whether there is an empty or non-empty relationship among two subscriptions. Hence we develop two additional closeness metrics, *Intersect-Over-Sum* (IOS) and *Intersect-Over-Union* (IOU), that not only account for both conditions and yield zero when two subscriptions have empty relationship, but also favor clustering higher traffic subscriptions (by taking the square of $|S_1 \cap S_2|$) since their impact on load is more significant than lower traffic subscriptions. In our evaluation, we test and compare the effectiveness of all four metrics.

We will now explain the foundation of the CRAM algorithm and then present optimizations to improve its runtime efficiency and clustering effectiveness. On initialization, CRAM allocates the subscriptions without any clustering using the BIN PACKING algorithm. If this fails, (i.e., due to insufficient broker resources) then the algorithm terminates. Otherwise, record the subscription allocation scheme. Then, CRAM repeatedly executes the following steps:

1) Find and cluster the pair of subscriptions having the next highest non-zero closeness value. When two subscriptions are clustered, their bit vectors are aggregated together using the OR bit operation to form a new subscription (see Figure 1). The subscription pool is updated of the new pairing. If no pairing can be found, then the algorithm ends and returns the last successful allocation scheme.
2) Allocate the subscriptions in the subscription pool using the BIN PACKING algorithm. If allocation fails, then undo and note the clustering (so that this subscription pair will not be paired again) and revert the changes made to the subscription pool. Otherwise, record the successful subscription allocation scheme.
3) Repeat step 1.

The purpose of recording the last successful allocation scheme upon initialization and at the end of step 2 is so that the algorithm can return an allocation scheme if all subsequent clustering results are not allocatable. In terms of computational complexity, the CRAM algorithm as we have described so far is O(S^3log(S)) because step 1 finds up to O(S^2) pairs of subscriptions, takes O(S^2log(S)) time to sort the subscription pairs in descending closeness, and repeats up to S iterations.

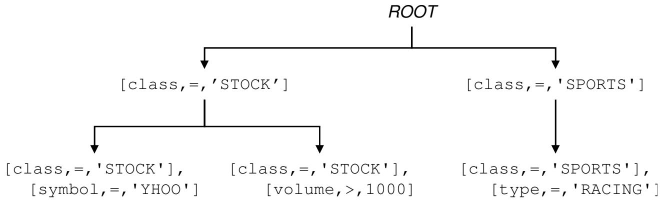


Fig. 2. The poset data structure

1) Optimization 1 - Grouping of Equal Subscriptions: In order to reduce the computation complexity of step 1, we need to reduce S, the number of subscriptions, by putting subscriptions with equal bit vectors into the same *Group of Identical Filters* (GIF). In our experiments using 8,000 subscriptions, S is reduced by up to 61%. Therefore, instead of clustering pairs of individual subscriptions, this optimization clusters multiple subscriptions from a pair of GIFs. However, the subscriptions to cluster depend on the relationship among the two GIFs. The relationship between a pair of GIFs is determined by the publications that they receive, which is captured by the bit vectors. If their relationship is equal, then the GIF is paired with itself. In this case, we use binary search to repeatedly find the largest allocatable cluster(s) of the GIF's subscriptions. If their relationship is intersect, such as the relationship of S_1 and S_2 illustrated in Figure 3, then we cluster the smallest loaded subscription from each GIF since the larger subscriptions within each GIF are clustered to their largest allocatable size already. If one GIF's bit vector is a superset or subset of another, then we cluster the lightest loaded subscription unit from the covering GIF with as many subscriptions from the covered GIF (by doing binary search on the set of covered subscriptions sorted in ascending order of output bandwidth requirement).

2) Optimization 2 - Search Pruning: Our second optimization is to limit the search space for finding the closeness between GIFs. Specifically, we want to avoid spending any CPU cycles on calculating the closeness between GIF pairs that have empty relationship or lower closeness than what is already found. At the same time, we also want to reduce the number of candidate GIF pairs to sort in step 1 from S^2 to S by keeping track of only the closest partnering GIF pair for each GIF. This effectively reduces the sorting complexity on each iteration of the algorithm from O(S^2log(S)) to O(Slog(S)), which reduces the complexity of the overall algorithm from O(S^3log(S)) to O(S^2log(S)).

In order to support our second optimization, we utilize the poset data structure [9] because it allows us to exploit an important property related to the closeness metrics, which we explain shortly. The poset is a directed acyclic graph where each unique subscription is represented as a node in the graph as shown in Figure 2. Nodes can have parent and children nodes where parent nodes have a subscription space that is a superset of its children nodes, while subscriptions with intersection or empty relationships will appear as siblings. The overhead of using the poset data structure is well worth the tradeoff too because the computational complexity for insertions and deletions is O(S). However we expect insertions and deletions to take on average O(log(S)) for a balanced poset structure. Our experiments support this claim as inserting 3,200 GIFs takes only 2 s.

Upon invoking the CRAM algorithm, each GIF is inserted as a node in the poset. In this work, we use the bit vectors from the subscription profiles to identify the relationship among GIFs instead of the subscription language as is commonly used in literature. The algorithm for identifying the relationship among subscriptions with one or more bit vectors is explained in our online Appendix [30]. With the poset initialized, CRAM iteratively takes a GIF from the GIF pool and calculates the GIF's closeness with each poset node, starting from the root node and iterating over the poset using breadth-first enumeration. Depending on which closeness metric is used, the enumeration ending criteria is different.

For the INTERSECT, IOS, and IOU metrics, the closeness value will either be zero (if GIF pair has empty relation) or a non-zero value (if GIF pair has non-empty relation). For GIF pairs with zero closeness, all of their descendent nodes are guaranteed to have zero closeness as well, which enables CRAM to prune the search process. For GIF pairs with non-zero closeness, the closeness value either remains constant (INTERSECT) or increases (IOS and IOU) traversing down the poset. The closeness value then starts to decrease when traversing past the GIF's own poset node where the number of intersecting bits start to decrease. Therefore, the search for a GIF's closest pair can be pruned once the closeness value starts to decrease, which is at the closest pair's children nodes. For GIF pairs that fail the allocation test, they need to be noted so that they will be skipped over in subsequent searches. Our experiment results demonstrate that finding the closest pair for every GIF using this optimization reduces the number of closeness computations from approximately 5,000,000 (assuming that the first optimization reduced 8,000 subscriptions to 3,200 GIFs) to 280,000. This means that in the average case, the computational complexity to search for the closest pair for each GIF is reduced from O(S) to O(log^2(S)).

The XOR metric [10], however, does not have the advantage of pruning the search over GIFs having empty relationships because the value of this metric is non-zero regardless of whether the relationship is empty or not. In fact, experiment results show that GIFs with empty relationships do actually get clustered together. Compared to the other three closeness metrics, XOR requires at least 75% longer computation time.

3) Optimization 3 - One-to-Many Clustering: The CRAM algorithm as we have described so far tries to cluster two GIFs on each iteration of the loop. This is similar with the pairwise subscription clustering algorithm in [24]. However, we develop an additional subscription clustering strategy that further improves on the effectiveness of pairwise clustering using the INTERSECT, IOS, and IOU metrics.

Consider the scenario illustrated in Figure 3. If we take each block in the grid as one bit, then subscription S_1 on the left has 36 bits and subscription S_2 on the right has 16 bits. The shaded 8 bits in the middle is $S_1 \bigcap S_2$, which is the set of publications that both S_1 and S_2 sink. Using the IOS metric as an example (INTERSECT and IOU will yield the

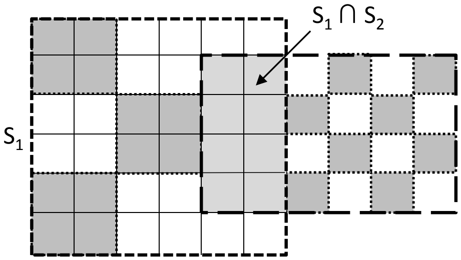


Fig. 3. One-to-many clustering approach of optimization 3

same verdict), the closeness between S_1 and S_2 is $8^2 \div 60 \approx 1.07$, between S_1 and any one of its covered subscriptions (2x2 shaded blocks) is $4^2 \div 40 = 0.4$, and between S_2 and any one of its covered subscriptions (1x1 shaded blocks) is $1^2 \div 25 \approx 0.04$. Since the closeness between S_1 and S_2 is highest, the pairwise algorithm will cluster S_1 and S_2 over clustering either S_1 or S_2 with their covered subscriptions. However, it is actually more favorable to first cluster S_1 (likewise for S_2) with its set of covered subscriptions before clustering S_1 with S_2 together because there is higher overlap between each of S_1 and S_2 with their covered subscriptions. The IOS metric also supports this claim as the closeness between S_1 (S_2) and all of its covered subscriptions is $12^2 \div 48 = 3$ ($8^2 \div 32 = 2$), which are greater than the closeness between S_1 and S_2 themselves.

Therefore, our 3rd optimization to CRAM is to first attempt to cluster each GIF with their covered GIFs if a candidate GIF pair has an intersect relationship. The lookup for the set of covered GIFs takes O(1) with the poset. The set of covered GIFs to cluster with the candidate GIF, called *Covered GIF Set* (CGS), should be chosen such that the GIFs in CGS have least overlapping set bits with each other to minimizing the number of GIFs to cluster (which increases the chance of passing the allocation test) while maximizing the total bit coverage (which increases the closeness value of the CGS with its parent GIF). Naturally, the greedy solution to the set cover problem fits in nicely with these requirements by mapping the ”1” bits within the bit vector(s) of each GIF in our approach to the elements of each set in the generic solution. That is, we maintain a list of the covered GIFs sorted by the number of bits not already in the CGS in descending order. On each iteration of the loop, include the next GIF in the head of the list to the CGS, update and re-sort each GIF remaining in the list according to the new CGS, and terminate when clustering any further GIF will force the CGS-parent cluster to exceed the load requirements of the original GIF pair. The intuition behind this terminating condition is to ensure a fair closeness value comparison. Once the CGS is finalized, the CGS is valid if it passes the allocation test and its closeness with its parent GIF is higher than the closeness between the original candidate GIF pair.

V. Phase 3: Broker Overlay Construction

In Phase 2, we have described three subscription allocation algorithms which allocate subscriptions to a set of brokers. In Phase 3, we design a tree overlay to connect the brokers allocated in Phase 2 with the assumption that the publishers will initially relocate to the root of the tree. Once the overlay is designed, CROC informs each broker to execute the reconfiguration, after which GRAPE strategically relocates the publisher clients on the final overlay.

Our goal in designing the broker overlay construction algorithm is to make it as least complex as possible while still being able to make systematic and intelligent decisions. To do so, we simply map each broker allocated in the previous run of the subscription allocation algorithm to a subscription profile by aggregating all subscription bit vectors serviced by the broker using the OR bit operator (similar to building a new subscription from clustering two subscriptions as illustrated in Figure 1) and then recursively invoke the subscription allocation algorithm. Pictorially, we are building the tree overlay layer-by-layer, starting with subscriptions at the bottom layer, with lesser-and-lesser brokers allocated at each higher layer until only one broker is allocated. That last allocated broker is the root of the tree and is also where all the publishers connect to in the new topology.

The benefits of our broker overlay construction algorithm is two-folds. One, we are reusing the subscription allocation algorithm in Phase 2, which means we are reusing as much code and logic as possible. This not only helps minimize code development time but also minimizes additional complexity into the algorithm. Two, the recursive nature makes the entire allocation scheme consistent among subscriptions and brokers. For example, if CRAM is used to allocate subscriptions to brokers, then CRAM is also used to build the broker overlay. Additionally, we introduce three new optimizations to the broker overlay construction algorithm that further reduce the number of brokers allocated in Phase 3. These optimizations are applied in the order that they are presented below after allocating each layer of brokers, which is just prior to the recursive invocation.

A. Optimization 1 - Eliminate Pure Forwarding Brokers

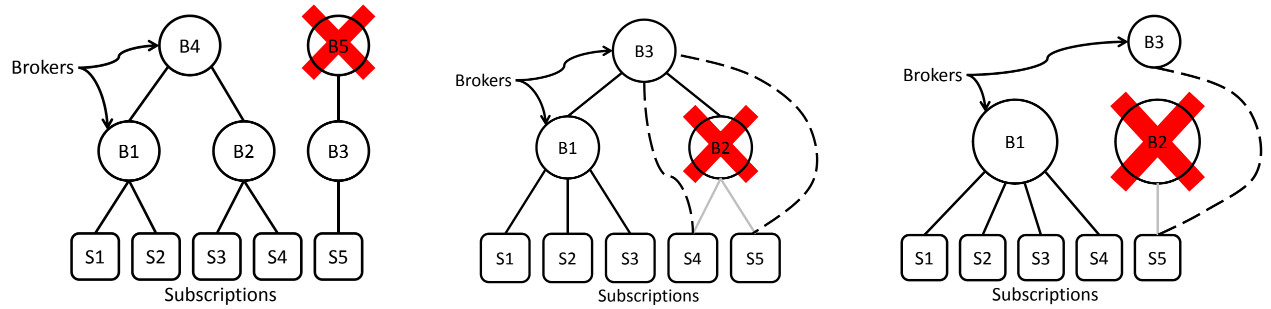
Figure 4a shows an example of a pure forwarding broker, *B5*, that has only one neighbor to forward publication traffic. Since pure forwarding brokers are not even multicasting and just purely forwarding incoming traffic, they can be safely deallocated to avoid that extra broker hop of unnecessary filtering and forwarding.

B. Optimization 2 - Takeover Children Broker Roles

Since our overlay construction algorithm allocates brokers layer-by-layer, it is possible that the load assigned to the last broker at each layer be very low. In this case, brokers allocated on the next higher level can potentially have the capacity to directly handle the load of the under-utilized child broker. Figure 4b demonstrates an example of such a scenario. To handle this anomaly, we check if any allocated broker in the last allocation run can directly handle the load of its children brokers in order of least-to-highest utilization. This sorting order maximizes the number of children brokers that can be taken over by the parent broker.

C. Optimization 3 - Best-Fit Broker Replacement

Also due to the same condition as described in the above section, allocated brokers may be under-utilized. Under-utilized resources are not only inefficient but can also lead



(a) Opt. 1: Deallocate B5 since it is a pure forwarding broker

(b) Opt. 2: B3 has enough capacity to serve B2's subscriptions directly along with B1

(c) Opt. 3: Replace under-utilized high resource brokers with low resource brokers

Fig. 4. Broker overlay construction optimizations

to more brokers allocated than is necessary. To deal with this problem, this optimization replaces each allocated broker with a broker whose resource capacity has the best-fit. Figure 4c illustrates an example of this optimization scheme.

VI. Evaluation

The main objective of our evaluation is to examine the performance gains and tradeoffs of our key contributions under homogeneous and heterogeneous environments: (1) FBF, BIN PACKING, and CRAM subscription allocation algorithms, (2) INTERSECT, IOS, and IOU closeness metrics in our CRAM algorithm, (3) recursive broker overlay construction algorithm, and (4) effectiveness of the bit vector based resource allocation framework. We also compare our work with the following two baseline and three related approaches: (1) two baseline subscription allocation and broker overlay construction approaches called MANUAL and AUTOMATIC (described below), (2) two derivatives of the pairwise clustering approach proposed in [24] which we refer here as PAIRWISE-K and PAIRWISE-N (described below), and (3) the Gryphon-derived closeness metric, XOR [10], in our CRAM algorithm.

One of our baseline approaches is called MANUAL, which forms the initial overlay topology that we use for all evaluations. In this approach, the broker overlay has a fan-out of 2 to minimize the chance of overloading internal brokers in the tree, and publishers are randomly placed on the overlay. Under the homogeneous scenario, subscribers too are randomly placed on the overlay. However, under the heterogeneous scenario, the most resourceful brokers are manually placed at the top of the tree and the number of subscribers allocated to each broker is proportional to the brokers' resource levels. Our other baseline algorithm is called AUTOMATIC which positions the clients and builds the broker overlay randomly. We chose to evaluate these two baseline approaches because they are representative of typical publish/subscribe deployments where the measure of a ”good” topology is not easily quantifiable [7], [8], [9], [10].

Moreover, we compare our work against two derivatives of the pairwise cluster algorithm both using the XOR closeness metric from [24]. The pairwise algorithms are derived because of two reasons. One, they originally do not allocate subscriptions to brokers nor build the broker overlay. Thus, we extend both pairwise algorithms to build the broker overlay using the AUTOMATIC approach. Two, we extend the pairwise algorithms to use bit vectors instead of the subscription language due to limited development time. Doing so actually benefits the related approaches because the workload that we use are stockquotes, which do not follow any well-defined distribution pattern. Thus, the use of bit vectors actually enable the pairwise algorithms to make better clustering decisions.

One of the derivatives is called PAIRWISE-K where we set the number of clusters to that computed by CRAM with the XOR closeness metric and then randomly assign subscription clusters to brokers. The reason for choosing the XOR closeness metric is because that is the metric used in [24]. The other derivative is called PAIRWISE-N where we set the number of clusters to the number of brokers in the system and assign each subscription cluster to a broker.

A. Experiment Setup

All three phases of our approach are implemented on top of PADRES [22], an open source distributed content-based publish/subscribe system developed by the Middleware Systems Research Group (MSRG) at the University of Toronto. The CBC is implemented into the PADRES broker as an additional internal component with 1,600 lines of Java code. CROC is implemented as a separate publish/subscribe client with 4,200 lines of Java code. Regarding the actual reconfiguration at the end of Phase 3, we re-instantiate every broker in the system and have the original clients connect to the new broker instances. We chose this method over reusing the original broker instances because this is the simplest and surest way to start brokers from a clean state.

We evaluated all baseline, related, and proposed approaches on two testbeds which mimic enterprise-scale data center environments [1]. One is a cluster testbed consisting of 21 nodes each with two Intel Xeon 1.86 GHz dual core CPUs connected by 1 Gbps network links. The other is SciNet, a high performance computing cluster consisting of 3,780 nodes each with two Intel Xeon 2.53 Ghz quad core CPUs connected by 1 Gbps network links. Deployments on both testbeds are aided by the use of a tool developed by MSRG called *PADRES Automated Node Deployer and Administrator* (PANDA). This tool allows us to specify the experiment setup within a text formatted *topology file* such as the time and nodes at which to run brokers and clients, as well as any process specific runtime parameters such as the neighbors for brokers. The topology file is fed into PANDA which then deploys the processes automatically. Brokers and overlay links are verified to be up and running before clients are deployed. A publisher and its set of matching subscribers run on the same machine to accurately measure end-to-end publication delivery delay. Different publishers run on randomly chosen machines that also run broker processes. Each publisher publishes stock quote publications of a particular stock that are real-world values obtained from Yahoo! Finance containing a stock's daily closing prices.[2] We chose stockquotes as the workload not only because it is real world data but it also highlights how our solution can cope with workloads of arbitrary distribution. A typical stockquote publication looks like this:

```
[class,‘STOCK’],[symbol,‘YHOO’],[open,18.37],
    [high,18.6],[low,18.37],[close,18.37],
    [volume,6200],[date,‘5-Sep-96’],
    [openClose%Diff,0.0],[highLow%Diff,0.014],
    [closeEqualsLow,‘true’],[closeEqualsHigh,‘false’]
```

We evaluated all approaches under both homogeneous and heterogeneous scenarios on the cluster testbed. Both scenarios consist of 80 brokers with 40 publishers each publishing a unique stockquote at 70 msg/min. Brokers in the homogeneous scenario all have equal processing and bandwidth capacities. As well, each publisher has an equal number of subscriptions. To see the impact of the number of subscriptions, we vary this number from 50 to 200 per publisher in increments of 50 on the cluster testbed. Thus, the total number of subscriptions ranges from 2,000 to 8,000 in increments of 2,000. Furthermore, we conducted large-scale deployments with 400 and 1,000 brokers with 225 subscribers per publisher on SciNet under a homogeneous setting. The total number of publishers are set to 72 and 100 for networks of sizes 400 and 1,000, respectively, to initially saturate the system which is represented by the baseline MANUAL case.

In the heterogeneous scenario, 15 brokers are configured to have 100% of the network capacity of the brokers in the homogeneous scenario, 25 brokers are configured to have 50% of the network capacity, and 40 brokers are configured to have 25% of the network capacity. We achieve bandwidth throttling through the use of a bandwidth limiter in each broker. The number of subscriptions for the i_{th} publisher is $N_s \div i$ with N_s ranging from 50 to 200 in increments of 50 per experiment. For example, in an experiment with N_s set to 200, the total number of subscriptions is 4,100, and the lowest and highest number of subscribers for a publisher are 5 and 200, respectively.

Using the YHOO stock as an example to describe our subscription workload, 40% of the subscriptions subscribe to the template [class,=,‘STOCK’],[symbol,=,‘YHOO’], while the other 60% also subscribe to that same subscription but with an additional inequality attribute, such as [class,=,‘STOCK’],[symbol,=,‘YHOO’],[low,<,REPLACE_LOW]. The value for that additional attribute is chosen randomly from the same field in that stock's publication set.

Since one of the main objectives of this work is to reduce the overall broker message rate, we configure GRAPE to focus 100% on minimizing the system load rather than the delivery delay. GRAPE's publication sample count was set to 1,000 to allow us enough time to capture the stabilized system state in between publisher relocations.

B. Experiment Results

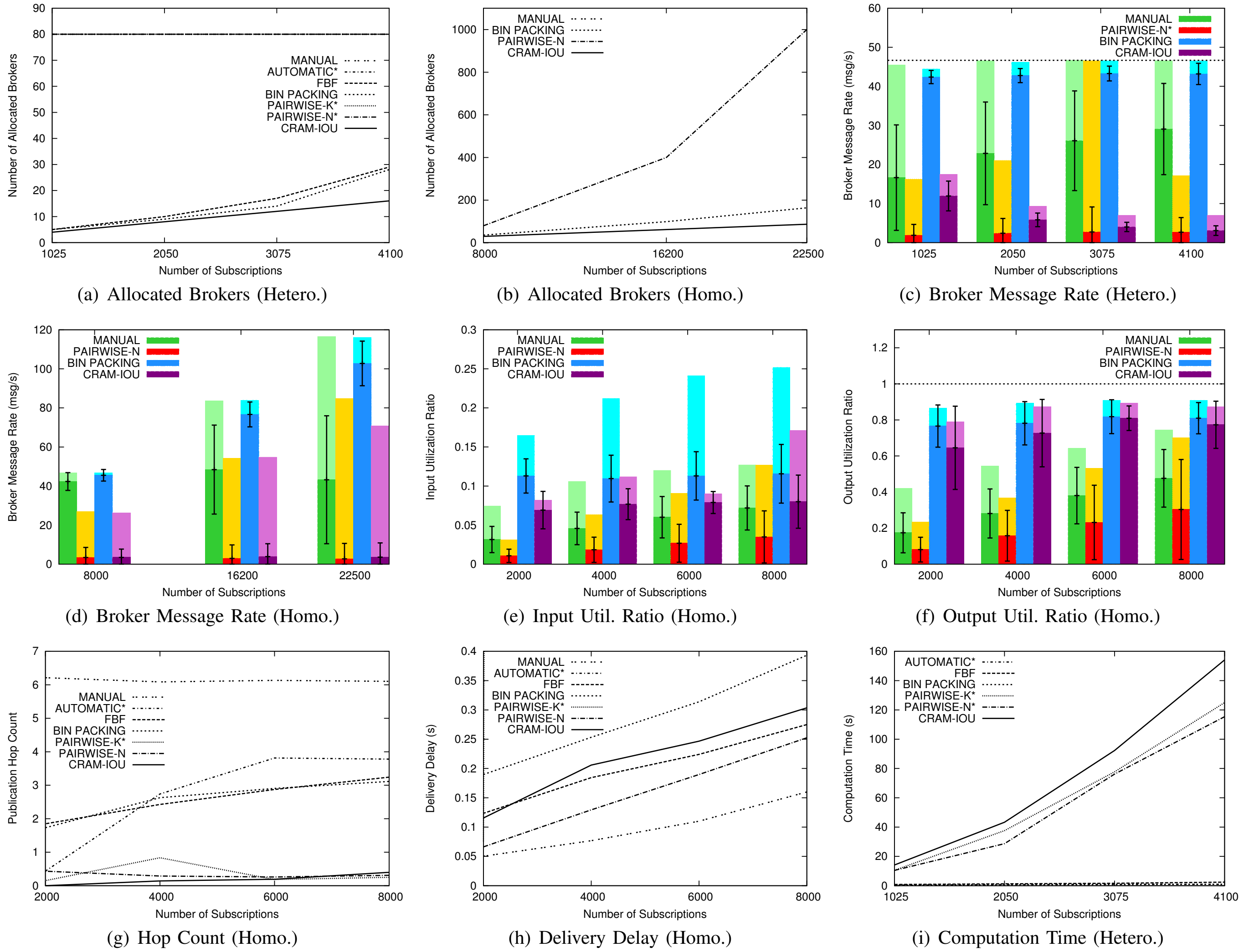
We will first show the results of our macro experiments which compare the performance among the best of the baseline approaches (MANUAL), related approaches (PAIRWISE-N), sorting algorithms (BIN PACKING), and closeness metrics (IOU) with the CRAM algorithm. Then, we analyze the gains by applying publisher relocation and subscriber relocation with overlay reconfiguration separately. Following that, we examine the results of our micro experiments comparing the two baseline approaches, two related approaches, two sorting algorithms, and four closeness metrics in the CRAM algorithm. Finally, we study the performance implications on varying the length of the bit vector. Our evaluation focuses on the following metrics: number of allocated brokers, resource utilizations, broker message rates, publication delivery delays and hop counts, and computation time of the algorithm.

The plotted values are an average over ten 1-minute intervals of the system at its converged state. All graphs use average values across all clients and *only* allocated brokers in the system. Most of our results are shown in this paper, with the full set available in an online Appendix [30]. Nevertheless, we summarize all of our results here in this paper. Algorithms labeled in the graphs with an asterisk denote that those algorithms lead to overloaded brokers and anomalies for them are expected due to erratic behavior of the JVM running out of memory.

1) Comparison of Baseline, Related, and Proposed Approaches: Figure 5a shows the number of brokers allocated over a range of subscription workloads on the cluster testbed under the heterogeneous scenario. Figure 5b shows the number of brokers allocated on the cluster (8000 subscriptions) and SciNet (beyond 16200 subscriptions) testbeds under the homogeneous scenario. In these graphs, lesser brokers allocated denote higher resource usage efficiency. Algorithms aware of broker capacities including BIN PACKING, FBF, and CRAM allocate less than half the total set of brokers. The number of brokers allocated also scales with the number of subscriptions in the workload. Both the baseline and related approaches always allocate all of the brokers because they lack awareness of broker capacities. Yet, both AUTOMATIC and PAIRWISE-K cannot avoid overloading at most one broker in the system. This emphasizes the need for resource awareness in allocation algorithms. CRAM-IOU consistently allocates the fewest number of brokers, with a reduction of up to 91% and 80% in the homogeneous and heterogeneous scenarios, respectively. Compared to BIN PACKING, CRAM-IOU allocates up to 47% less brokers than BIN PACKING. Yet, BIN PACKING always allocates one less broker than FBF which is inline with theoretical expectations [29]. A similar trend is observed in the homogeneous scenario.

Figure 5c shows the average, standard deviation, and maximum broker message rates under the heterogeneous scenario. Figure 5d shows the same metrics but on the cluster

[2] Available at http://www.msrg.org/datasets/acDataSet



(a) Allocated Brokers (Hetero.)

(b) Allocated Brokers (Homo.)

(c) Broker Message Rate (Hetero.)

(d) Broker Message Rate (Homo.)

(e) Input Util. Ratio (Homo.)

(f) Output Util. Ratio (Homo.)

(g) Hop Count (Homo.)

(h) Delivery Delay (Homo.)

(i) Computation Time (Hetero.)

Fig. 5. Experiment results

(8000 subscriptions) and SciNet (beyond 16200 subscriptions) testbeds under the homogeneous scenario. The lower is the broker message rate, the more efficient are resources being used. Compared to MANUAL, PAIRWISE-N shows that clustering subscriptions significantly reduces both the average and maximum broker message rates while still allocating all 80 brokers. However, PAIRWISE-N suffers from broker overloads in the heterogeneous scenario because it has no resource awareness. BIN PACKING, a simple sorting algorithm with resource awareness, allocates less than 88% of the brokers in the network. However, without clustering, brokers sink almost all of the publications sent by all publishers in the system, which is represented by the horizontal dotted line at 46.7 msg/s in 5c. With subscription clustering and broker capacity awareness, CRAM-IOU combines the benefits of low message rates from PAIRWISE-N and low number of allocated brokers from BIN PACKING without overloading any brokers. Figure 5d shows that CRAM-IOU reduces the average and maximum broker message rates by up to 92% and 85%, respectively. As well, PAIRWISE-N did not experience any overloads in the homogeneous scenario.

Figure 5e shows the average, standard deviation, and maximum broker input utilization ratios. Input utilization ratio captures the brokers' matching rate versus the flow of incoming traffic[1]. Because the input utilization ratio varies directly with the broker message rate, the trend in Figure 5c is carried over to this graph as well. BIN PACKING, with highest broker message rate, imposes highest average and maximum input utilization on the brokers. CRAM-IOU's input utilization is higher than PAIRWISE-N's because CRAM-IOU allocates significantly less brokers and every allocated broker in CRAM-IOU is fully utilized in terms of output bandwidth, whereas the brokers in PAIRWISE-N are not.

Figure 5f shows the average, standard deviation, and maximum broker output utilization ratios. Output utilization ratio captures the output bandwidth usage of a broker[1]. Since the broker's output resource is the bottleneck, the goal is to saturate the output bandwidth as much as possible to achieve maximum resource usage efficiency without overloading the broker. Algorithms that are aware of broker capacities, namely BIN PACKING and CRAM-IOU in this graph, are effective at using up on average 80% of the output resource regardless of the workload and scenario. Also notice that all allocated

[1] Please see [11] for definitions of these metrics

brokers in BIN PACKING and CRAM are much better utilized as demonstrated by the small standard deviation and difference between the maximum and average values as compared to the baseline and related approaches. The output utilization ratio of PAIRWISE-N is lower than MANUAL because of the significant reduction in broker message rates as shown previously in Figure 5c. Results from the heterogeneous scenario follow a similar trend except that PAIRWISE-N suffers from output overload as we will show later in Figure 6f.

Given that both CRAM-IOU and BIN PACKING allocate fewer brokers and dynamically construct the broker overlay, the network size is expected to be much smaller. This fact is supported by Figure 5g which shows the average publication hop count. Lower hop count is favorable because the publication delivery path is shorter, which reduces the amount of resources used in processing publications. In this graph, MANUAL is expected to be the highest because the maximum length of the network is 11 with 80 brokers and a fanout of 2. Also using 80 brokers but with a higher fanout, AUTOMATIC achieves lower hop count than MANUAL. BIN PACKING yields even lower hop count thanks to significantly smaller network size made up of much less brokers. CRAM-IOU's hop count is lowest of all approaches thanks to both subscription clustering and small network size. A similar trend is observed in the heterogeneous scenario where CRAM reduces the hop count to 0 across all workloads.

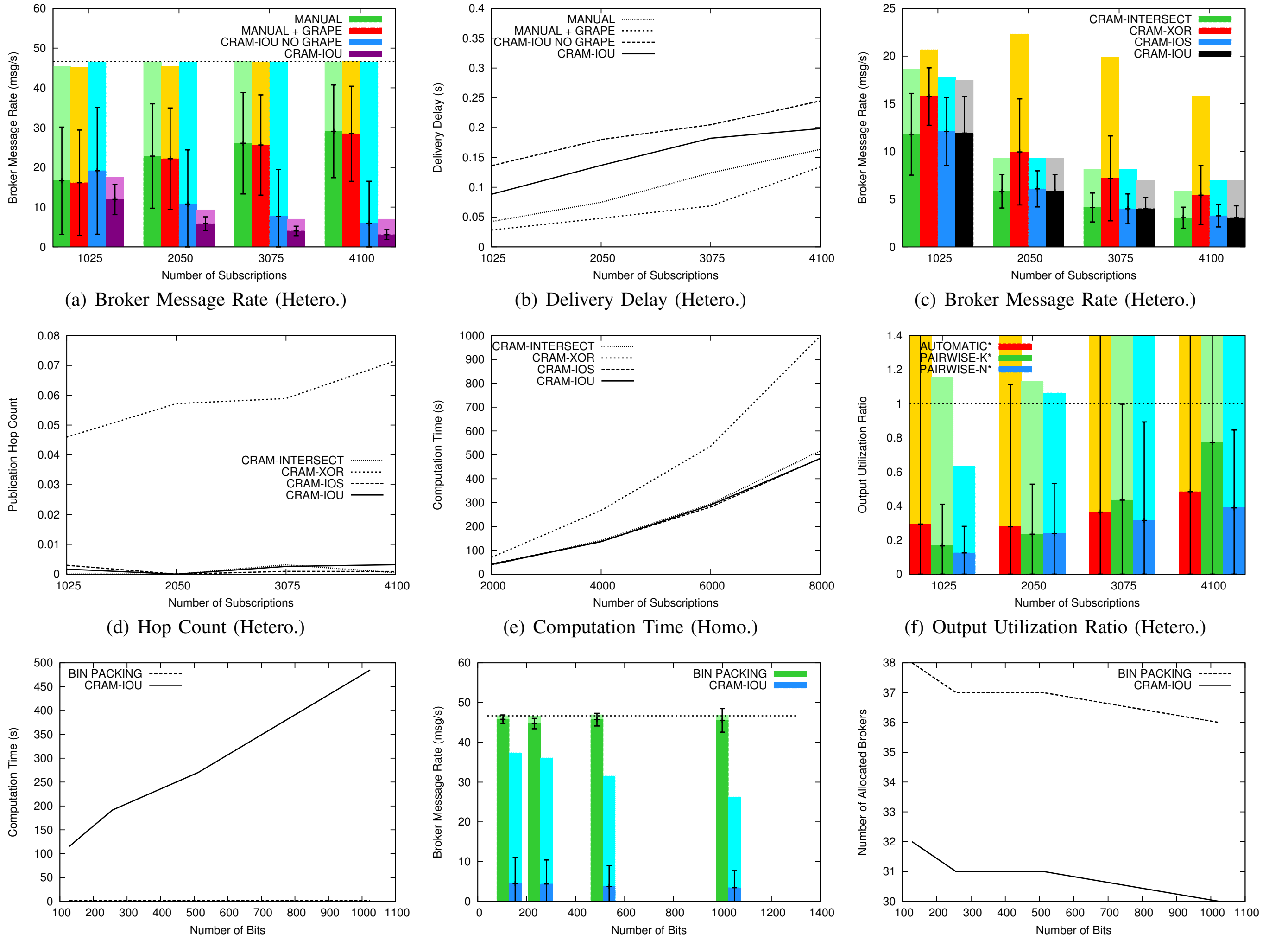
Contrary to common believe, reduction to hop count does not always translate to lower delivery delay as shown in Figure 5h. In a real system, the delivery delay also depends on the contention for the broker's output bandwidth. Which means, the more spread out are the subscribers of common interests, the lower is the bandwidth contention at the broker during publication delivery. This observation is supported by the delivery delay shown for the MANUAL case under both scenarios. Using the same number of brokers as MANUAL but with subscription clustering, publications matching all N subscriptions have to be queued up at one broker to be delivered to the N subscribers, which increases the delivery delay for PAIRWISE-N by as much as 56% compared to MANUAL. This also explains why the delivery delay for BIN PACKING is higher than FBF because all heavy traffic subscriptions in BIN PACKING are allocated to the same brokers. Still, the delivery delay of CRAM-IOU is below that of BIN PACKING, which we believe is due to the significant reduction in broker message rate as shown previously in Figure 5c. The delivery delays for AUTOMATIC and PAIRWISE-K are well beyond the range of this graph due to overloaded brokers. In the heterogeneous scenario, the delivery delays among all of the approaches are smaller because of fewer number of subscriptions per publisher, which reduces the bandwidth contention. However, PAIRWISE-N suffers from high delivery delays in the heterogeneous scenario due to overloaded brokers.

Figure 5i shows the time required to compute the subscription allocation and broker overlay, where lower is obviously better. The computation time for AUTOMATIC and the sorting algorithms is the shortest and grows linear with the number of subscriptions. CRAM-IOU takes the longest to run because it has to invoke the BIN PACKING algorithm after clustering every candidate GIF pair. The pairwise algorithms take approximately 15% and 25% less time than CRAM-IOU in the homogeneous and heterogeneous scenarios, respectively, mainly because these algorithms do not recursively cluster brokers nor invoke the BIN PACKING algorithm after clustering each GIF pair. Both CRAM and the pairwise algorithms' computation times are shown to vary directly with the squared of the number of subscriptions. Later in this paper, our micro experiments show that the computation for CRAM-IOU depends linearly on the length of the bit vector.

2) Breakdown of Performance Gains: Figure 6a shows the broker message rates of MANUAL and CRAM with and without publisher relocation. Starting off with MANUAL + GRAPE, results show that in a system where subscriptions are randomly placed in the network, the message rate cannot be reduced by simply relocating publishers. We believe that strategies that only reconfigure the broker overlay [7], [12], [13] also have the same effect under such a scenario. Clustering subscriptions and allocating them on a new broker overlay reduces the average broker message rate by up to 90% in the homogeneous scenario. However, without adaptively relocating the publisher, the maximum message rate is unchanged from MANUAL + GRAPE. It is only until after we manipulate all three variables, subscription placement, broker overlay connections, and publisher placement, that we are able to reduce the maximum message rate by up to 85% in the heterogeneous scenario. Not shown, the maximum message rate of CRAM-IOU in the homogeneous scenario is higher than the heterogeneous scenario because not all subscriptions for a publisher can be serviced at the same broker. Additionally, publisher relocation slightly reduces the input utilization ratio of CRAM-IOU, but has no impact on the output utilization ratio.

From the client's perspective, relocating publishers reduces the hop count on average by 40% in both scenarios [30]. In the heterogeneous scenario, publisher relocation actually reduces the hop count of CRAM-IOU to zero. According to Figure 6b, relocating publishers reduces the delivery delay on average by 16% in the homogeneous case and 24% in the heterogeneous case. Therefore, one can view publisher relocation as a remedy to reduce the delivery delay penalty for using CRAM to maximize the efficiency and minimize the allocation of resources.

3) CRAM Closeness Metrics: Figures 6c to 6e show the broker message rate, publication hop count, and computation time, respectively. According to these graphs and [30], the average and maximum message rates for XOR are on average 55% and 93% higher than IOU, respectively. This is because the XOR metric clusters subscriptions with empty relationships together. The fact that XOR produces 74% (41%) fewer clusters than IOU in the homogeneous (heterogeneous) scenario supports this claim. Another notable difference is that XOR yields 2,200% higher hop count and 18% higher delivery delay compared to the IOU metric. XOR also requires 105% (75%) more computation time than all other closeness metrics in the



(a) Broker Message Rate (Hetero.)

(b) Delivery Delay (Hetero.)

(c) Broker Message Rate (Hetero.)

(d) Hop Count (Hetero.)

(e) Computation Time (Homo.)

(f) Output Utilization Ratio (Hetero.)

(g) Computation Time (Homo.)

(h) Broker Message Rate (Homo.)

(i) Allocated Brokers (Homo.)

Fig. 6. Experiment results

homogeneous (heterogeneous) scenario. The primary reason for this is because of its inability to identify subscriptions with empty relation which the other closeness metrics have that enable them to prune the search for non-beneficial clusters.

INTERSECT, IOS, and IOU metrics show almost no differences in their input utilization ratios, output utilization ratios, number of allocated brokers, computation times, and number of subscription clusters. Initially, we expected the performance of the INTERSECT closeness metric to be inferior to IOS and IOU because of its inability to capture the extra traffic introduced in GIF pairs having intersecting relationships. However, the use of poset data structure to search for the next closest GIF pair eliminated this deficiency and enabled it to perform equally well with IOS and IOU.

4) Sorting Algorithms: According to the results shown in [30], the average input and output utilization ratios of BIN PACKING exceed FBF, which indicates that BIN PACKING is more effective in allocating more subscriptions into the brokers. We believe this is because BIN PACKING allocates subscriptions in the order of highest-to-lowest bandwidth requirement. As well, the maximum input utilization is higher in BIN PACKING because the sorting of subscriptions in descending traffic allocates more, if not all, high traffic subscriptions matching each publisher into the same broker. There are no significant differences in terms of the broker message rates among the two sorting algorithms.

5) PAIRWISE and AUTOMATIC Algorithms: Figure 6f shows that there exists overloaded brokers in AUTOMATIC and PAIRWISE-K under all workloads tested, and PAIRWISE-N when the number of subscriptions exceed 1,025 in the heterogeneous scenario. The main reason for the overload is allocating too many subscriptions to a broker and/or broker neighbors which exceed the broker's output bandwidth capacity. The take-away message here is the necessity of resource awareness in the subscription assignment and broker overlay construction algorithms.

6) Impact of Bit Count in the Bit Vector: Use of the bit vector scheme to predict traffic based on past data is very powerful in that it can adapt to workloads of any well-defined distribution. However, if the workload distribution changes, such as stockquotes in our experiments, then it is impossible to get 100% accurate estimation based on past data. To handle this scenario, a scaling factor is required to accommodate for potential additional traffic to prevent overloading brokers. With a bit vector length of 1,024 bits, we have to fix the minimum bits set to 20% and increase the bit count for subscriptions with less than 40% bits set by 25%. For bit vector lengths of 512 or less, we have to fix the minimum bits set to 35% and scale subscriptions with less than 25% and 40% traffic by 30% and 45%, respectively.

Figures 6g to 6i show the graphs for computation time, broker message rate, and allocated brokers over bit vector lengths ranging from 128 up to 1,024, respectively. The results show that the computation time varies linearly with the bit vector length. However, because the computation time for CRAM is much longer, we can see the linear effect more prominently with CRAM-IOU than BIN PACKING. For CRAM-IOU, increasing the bit vector length also increases the subscription load estimation accuracy, which in turn decreases the maximum broker message rates, increases the average input utilization ratio, and decreases the number of brokers allocated. This translates to better distribution of load and more efficient use of broker resources. As for BIN PACKING, the only notable difference by increasing the bit vector is fewer allocated brokers. Each reduction in an allocated broker increases the delivery delay, which is inline with our earlier observation in Figure 5h.

VII. Conclusions

The objective of this work was to find a subscription assignment, broker overlay topology, and publisher placement to use as few broker resources as possible given an arbitrary workload and a set of broker resources. The motivation behind our goal is inline with the current green IT initiatives in using resources as efficiently as possible, which not only decreases IT operational costs but also increases the capacity of the system by allowing it to handle more load. Our key contributions in this work include developing a bit vector supported resource allocation framework, designing and comparing four different classes with a total of ten variations of subscription allocation algorithms, developing a recursive overlay construction algorithm, and comparing our work against two baseline and three related approaches implemented on the PADRES [22] open source content-based publish/subscribe system. A compelling feature of our approach is that it works under any arbitrary workload distribution and is independent of the publish/subscribe language, which makes it easily applicable to any topic and content-based publish/subscribe system. Although the focus of this work was purely a centralized static approach, it can easily be integrated with existing distributed dynamic approaches that actively grow and shrink the network such as [31]. Future work can also examine ways to extend the current approach to be more fault tolerant.

Experiment results on a cluster and high performance computing testbed show that CRAM excels over both baseline and related approaches in many ways. One, CRAM reduces the average broker message rate by up to 92% whereas prior work on only relocating publishers [5] achieved no reduction whatsoever. Two, CRAM overcomes limitations in earlier work on subscription clustering [24] by not only overcoming broker overload issues and preventing excessive delivery delays but also reducing the number of allocated brokers by up to 91% and bringing the publication hop count down to zero. Three, our proposed clustering closeness metric, IOU, is capable of reducing the computation time of the CRAM algorithm by up to 105% over the Gryphon-derived closeness metric, XOR [10]. At the same time, we found tradeoffs in the CRAM algorithm including slightly longer computation time compared to prior subscription clustering approaches and higher delivery delays compared to baseline approaches. However, we also discover remedies to these tradeoffs through adjustments to the bit vector length to decrease computation time and strategically relocate publishers to reduce the delivery delay penalty.

References

[1] J. Reumann, “Pub/sub at Google,” CANOE and EuroSys Summer School, 2009.

[2] Tibco, “Tibco software chosen as infrastructure for nasdaq’s supermontage,” 2001. [Online]. Available: www.tibco.com

[3] GS1. [Online]. Available: http://bit.ly/cjnevk

[4] S. Enterprise, “Green IT report - United States and Canada,” 2009. [Online]. Available: http://bit.ly/a23yb7

[5] A. K. Y. Cheung and H.-A. Jacobsen, “Publisher placement algorithms in content-based publish/subscribe,” in *ICDCS’10*.

[6] N. Tajuddin, “Techniques for overlay design of content-based publish/subscribe systems,” Master’s thesis, U of T, 2010.

[7] R. Baldoni *et al.*, “Efficient publish/subscribe through a self-organizing broker overlay and its application to SIENA,” *TCJ*, 2007.

[8] S. Pallickara and G. Fox, “NaradaBrokering: a middleware framework and architecture for enabling durable peer-to-peer grids,” in *Middleware’03*.

[9] A. Carzaniga *et al.*, “Design and evaluation of a wide-area event notification service,” *ACM ToCS*, 2001.

[10] G. Banavar *et al.*, “An efficient multicast protocol for content-based publish-subscribe systems,” in *ICDCS’99*.

[11] A. K. Y. Cheung and H.-A. Jacobsen, “Dynamic load balancing in distributed content-based publish/subscribe,” in *Middleware’06*.

[12] M. A. Jaeger *et al.*, “Self-organizing broker topologies for publish/subscribe systems,” in *SAC’07*.

[13] M. Migliavacca and G. Cugola, “Adapting publish-subscribe routing to traffic demands,” in *DEBS’07*.

[14] P. Strong, “eBay - very large distributed systems (a.k.a. grids) @ work,” BEinGRID Industry Days, 2008.

[15] S. Ghemawat *et al.*, “The Google file system,” *SOSP’03*.

[16] G. Cugola *et al.*, “The JEDI event-based infrastructure and its application to the development of the OPSS WFMS,” *IEEE TSE*, 2001.

[17] T. Fawcett and F. Provost, “Activity monitoring: noticing interesting changes in behavior,” in *KDD’99*.

[18] B. Mukherjee *et al.*, “Network intrusion detection,” *IEEE Network*, 1994.

[19] G. Li *et al.*, “A distributed service-oriented architecture for business process execution,” *ACM Trans. Web*, 2010.

[20] M. Petrovic *et al.*, “G-ToPSS: Fast filtering of graph-based metadata,” in *WWW’05*.

[21] W. Yan *et al.*, “Efficient event-based resource discovery,” in *DEBS’09*.

[22] PADRES. [Online]. Available: www.msrg.org/projects/padres/

[23] L. Opyrchal *et al.*, “Exploiting IP multicast in content-based publish-subscribe systems,” in *Middleware’00*.

[24] A. Riabov *et al.*, “Clustering algorithms for content-based publication-subscription systems,” in *ICDCS’02*.

[25] T. Wong *et al.*, “An evaluation of preference clustering in large-scale multicast applications,” in *INFOCOM’00*.

[26] S. Voulgaris *et al.*, “Sub-2-sub: self-organizing content-based publish subscribe for dynamic large scale collaborative networks,” in *IPTPS’06*.

[27] J. A. Patel *et al.*, “Rappel: exploiting interest and network locality to improve fairness in publish-subscribe systems,” *Comp. Networks*, 2009.

[28] M. Altinel and M. J. Franklin, “Efficient filtering of XML documents for selective dissemination of information,” in *VLDB’00*.

[29] E. G. Coffman, Jr. *et al.*, *Approximation algorithms for bin packing: a survey*, 1997.

[30] A. K. Y. Cheung and H.-A. Jacobsen, “Appendix,” 2010. [Online]. Available: http://bit.ly/98MUXK

[31] Y. Yoon *et al.*, “Foundations for highly-available content-based publish/subscribe overlays,” in *ICDCS’11*.