# Using Dedicated and Opportunistic Networks in Synergy for a Cost-effective Distributed Stream Processing Platform

Shah Asaduzzaman and Muthucumaru Maheswaran Advanced Networking Research Lab School of Computer Science McGill University Montreal, QC H3A 2A7, Canada {asad, maheswar}@cs.mcgill.ca

# Abstract

This paper investigates how a hybrid hosting platform made from dedicated and opportunistic resources can be used to host data stream processing applications. We propose a system model for the hybrid hosting platform and develop resource management algorithms that are necessary to coordinate the allocation of the two classes of resources to the stream processing tasks. We used extensive simulations driven by traces styled from realistic system observations for evaluating the proposed resource allocation heuristics. The results show that with proper management, the synergy of dedicated and opportunistic resources yields considerably higher service throughput and thus, higher return on investment over expensive dedicated resources.

## 1 Introduction

Many applications on the Internet are creating, manipulating, and consuming data at an astonishing rate. Data stream processing is one such class of applications where data is streamed through a network of servers that operate on the data as they pass through them. Depending on the application, data streams can have complex topologies with multiple sources or multiple sinks. Examples of data stream processing tasks happen in many areas including distributed databases, sensor networks, and multimedia computing. Some examples include: (i) multimedia streams of real-time events that are transcoded into different formats, (ii) insertion of information tickers into multimedia streams, (iii) real-time queries on computer network monitoring data for malicious activitiy detection, and (iv) function computation over data feeds obtained from sensor networks.

One of the salient characteristics of this class of applica-

tions is the demanding compute and network resource requirements. In previous works, various researchers have examined the problem of mapping data stream processing tasks to distributed resource clusters. Due to their stringent quality-of-service requirements, these applications are mapped onto dedicated servers and networks so that the loading conditions on the resources can be controlled to provide the expected service rates. While dedicated server capacities are affordable, dedicated networks over wide-area installations remain costly. In this paper, we explore a novel approach where the network that interconnects server resources can include dedicated links and the public network. Our research explores how such a hybrid (called as *bi-modal* in this paper) network can be best used for data stream processing tasks. We develop various heuristic solutions for the resource management problem in this hybrid scenario and thoroughly evaluate the performance under representative settings.

This paper extends some of our previous work [3, 2] on bi-modal compute platforms where dedicated resource pools were augmented with opportunistically harvested public resources to increase service level compliances and the utilization of dedicated resources. Using data stream processing tasks as a concrete example, this paper demonstrates the benefit of using bi-modal network infrastructures for compute- and network-intensive applications. In particular, this paper makes the following contributions to this important resource management problem:

- Show that a bi-modal network can improve the utilization of dedicated resources such as servers and network links.
- Show that bi-modal network can improve the compliance levels of service contracts while admitting large workloads with minimal elongations in task durations.
- Show the importance of dynamic re-scheduling to

cope with changing loading conditions on the public network.

In Section 2 we present the system model for the data stream processing and the associated resource mapping problem. Section 3 discusses two types of resource mapping algorithms: an initial resource allocation and a dynamic re-scheduling algorithm. Section 4 examines the results from the extensive simulation studies we carried out to evaluate the algorithms. Section 5 reviews related literature.

### 2 System Model and Assumptions

## 2.1 Scenario

Consider a video stream task that can progress through several service components such as encoding, embedding of real-time weather or financial tickers and transcoding into different formats. Each of the individual services may be served by different servers in distributed locations. After being processed through all the steps, the stream may be delivered to a specific user. Figure 1 illustrates a scenario of a stream processing platform containing five servers (dedicated machines). Each server may serve multiple different service components as well as data sources. The servers are connected to the public Interent. In addition, some of the server pairs are conected using dedicated links. The example stream processing task shown in the figure requests a data stream from data source  $d_2$  to be processed through services  $a_2$ ,  $a_3$ ,  $a_4$  and  $a_5$ , and to be delivered to a host in the network  $N_1$ . This task may be served by the servers  $S_4$  (serving  $d_2$ ),  $S_3$  (serving  $a_2$ ),  $S_2$  (serving  $a_3$  and  $a_4$ ). Either dedicated link or public network may be used to transmit the data stream between any two consecutive servers.

# 2.2 Architecture

The stream processing platform can be viewed to be composed of the layers showed n Figure 2, with user applications at the top. The applications are composed of data sources and several service components hosted by differerent servers. Therefore, the service components constitue the middle layer. At the bottom layer, the resource management system (RMS) of the platform manages the available server and network resources to allow seamless execution of the service components. The main focus of this paper is to design and analyze the algorithms for various functionalities of the RMS layer. The RMS is responsible for mapping of the task requests on available resources and dynamically adapting the resource allocations in response to various loading conditions. The three components of RMS cooperate to achieve these functionalities. A detailed discussion on the RMS is presented in Section 3. RMS uses

the local operating system API to control the underlying resources. Hence host OS and physical resources lie at the bottom of the layered architecture.

# 2.3 Task Specification

The stream processing tasks are launched by users in the form requests for delivering some data stream from an origin to the user after several processing steps. In addition to the source, the request specifies the destination node, total volume of data to be extracted from the source, the series of the application types needed for processing of the stream and a required delivery data rate R. All these specifications forms the *service level agreement* (SLA) between the user launching the task and the platform. When the platform accepts the task after necessary resource allocation, it is responsible to meeting the constraints specified in the SLA.

Because each service type has its characteristic resource usage factor and bandwidth shrinkage factor, it is easy to calculate the CPU and link resource requirements for each hop of the stream. Given that many servers may serve any particular type of service component, it is the responsibility of the platform to map the requested components on appropriate nodes subject to fulfillment of the capacity constraints.

To monitor the compliance with the required specification, the SLA includes a monitoring time window T across which the delivery rate is measured for conformance. If the actual delivery volume in an measurement interval is V, the platform is compliant if  $V \ge RT$  and the platform is entitled to full price of the service. Otherwise, the platform is penalized at a rate proportional to RT - V

# **3** Decentralized Management of Server and Network Resources

The resource management system (RMS) of the stream processing platform has a completely distributed architecture, with an RMS agent in each of the server nodes. The agent on a particular server node is responsible for managing the computing resources that belongs to and the network links that originate from that node. Each RMS agent has three components - i) map manager ii) reservation manager iii) dynamic scheduler. The map managers of different RMS agents collaboratively maps a requested stream processing task specification to the available server and network resources. The reservation managers collaborate to reserve necessary resources based on one of the several alternative mappings generated by the map managers. The details of the mapping and reservation protocols and the underlying distributed algorithm to compute the optimal mapping is discussed in Section 3.1.



Figure 1. Example scenario of a data stream processing platform with five servers interconnected by the public Internet and dedicated links.

Figure 2. Layered architecture

The rate at which the data stream is processed and delivered to the target node does not remain constant even after acquiring the required amount of node and link resources successfully. Because, although the bandwidth of a dedicated link can be deterministically allocated and reserved by the source end of the link, when two servers are connected using public Internet paths, the end-to-end bandwidth cannot be deterministically allocated. To obtain maximum possible compliance with the service agreements in presence of such variability in the public network connections, the dedicated links and the public links need to be dynamically re-allocated based on the needs of the tasks. The dynamic scheduler handles this periodic reallocation of the different types of links. The details of the periodic re-allocation algorithm is presented in Section 3.2.

# 3.1 Mapping and Reservation

A user of the distributed platform uses one of the server nodes as a portal to launch its stream processing task. The task specification submitted to the portal contains the address of data stream source and an ordered list of the service components that should process the data stream. By default the delivery point (destination) of the stream is the user's portal node, but any other node can be specified as well. The specification also includes the required rate of data delivery, time window for monitoring the rate and pricing for each byte of data delivered.

After receiving the specification from user, the portal node engages the map manager component of its RMS agent to initiate the mapping of the specified requirements on the network. Through message passing among the map managers in different server nodes, the distributed mapping algorithm results in a set of feasible maps at the map manager of the data-source node. Each of the maps defines a path from the data source node to the delivery node through the server nodes that serve necessary service components. The best among the available feasible maps according to a certain cost metric is chosen and the reservation manager initiates the reservation of server and link resources.

The reservation protocol basically sends the reservation probe along the actual network path found in the map. The reservation manager in each server node along the path tries to allocate the node and link resources prescribed by the map. If the allocation is successful, it forwards reservation probe to the next server node in the map. In case the node finds that the required amount of resource is no longer available, it sends a reservation rollback message to the previous node in the path. Receiving a rollback message, the reservation manager releases the resources reserved for that particular task and forwards the rollback to the previous node in the map. Once the rollback message is received by the source node, it re-initiates the reservation on the next map in the priority queue. Once a successful reservation probe reaches the destination node, a confirmation is sent back to the data source node along the path and the data streaming begins.

#### 3.1.1 Distributed Algorithm for Mapping

The distributed mapping of the task specification on available resources is collaboratively performed by the map managers by gradually expanding the maps to neighbors in the server network. Receiving the task specification from the user, the portal server computes the amount of CPU and communication resources necessary for each component services. This is possible because each service type has well defined bandwidth shrinkage factor (ratio of output to input data rate) and CPU usage factor (CPU usage per input byte). The portal server then composes the initial map message, mapping the data delivery point on the desired server node and send the message to that server. Upon receipt of a *map* message, the map manager of each node invokes the *ProcessMap* algorithm (Algorithm 1).

### Algorithm 1 ProcessMap(u, m, T)

- 1: Map message *m* containing the mapping of first *j* services on a series of server nodes is received by node *u*. *j* is called the *prefix-length* of *m*. *T* denotes the ordered set of services in the task
- 2: **if** *u* is the data-source node and all the services except the data source is mapped in *m* **then**
- 3: m is a feasible map
- 4: **else**
- 5: **for** x = 0 to |T| j 1 **do**
- 6:  $m_x = \text{map}$  found by extending next x services in T on u
- 7: for each neighbor v of u that is not already in mdo
- 8: **if** available bandwidth in (u, v) link can support the bandwidth need for the service hop (j + x, j + x + 1) **then** 
  - Send  $m_x$  to v
- 10: **end if**

9:

- 11: end for
- 12: **end for**

13: end if

The distributed algorithm is based on a centralized algorithm, details of which including correctness and complexity analysis can be found at [1, 4]. Each node receiving the map message extend the map by mapping the next service on itself if the service is available and the CPU capacity allows, or acts as a forwarding node. The extended map is sent as a new map message to the neighboring servers if the available bandwidth permits. Cycles are avoided in these extensions. Eventually, the data source node receives a complete map of the task and initiates the reservation protocol. It follows from the correctness of the centralized algorithm [1] that the distributed mapping completes after at most N - 1 ProcessMap invocation by each node in the network (N is the total number of nodes). The algorithm terminates after all the outstanding ProcessMap have been completed. Since cycles are avoided during extension, an initial mapping may be extended to at most N - 1 hops. Thus there will be a finite number of *ProcessMap* invocation and the algorithm will terminate after a finite amount of time. After completion, the algorithm results in the set of all feasible mappings at the data-source node. Based on some desired cost-metric, the lowest cost mapping can then be selected.

The notion of neighborhood of a server node is straightforward when there is only dedicated links interconnecting the nodes. In case overlay links through the public Internet is allowed, each node can potentially connect to all other nodes in the network. However, it is only necessary to extend the maps to the nodes that serve the next service component in the task. We assume that an underlying gossip like algorithm disseminate the presence of services in each server across the network. Thus for each service type, each node has knowledge of a uniform subset of the nodes that hosts that service.

#### 3.1.2 Heuristic Approximations

As the mapping problem is NP-complete [4], computational complexity of both the centralized and the distributed path mapping algorithm grows exponentially with the problem size. Therefore, for practical deployment, we need some heuristic that produces good approximation to the optimal result. Here we discuss three possible heuristics that modifies the original algorithm to reduce computational, messaging and memory complexity.

### LeastCostMap

One major source of growth in complexity of the algorithm is the exponential growth of the set of partial maps extended by each node. In the *LeastCostMap* heuristic, each node maintains a table of the least cost map observed so far for partial maps of each possible prefix-length. If a new map is received, the cost of the new map is compared with that of the already stored one, and the map with higher cost is discarded. It is quite possible that the lower cost map that is selected will not eventually lead to a feasible solution, rather the dropped map would. Thus there may be no feasible mapping found when such map exists, or the optimal map may be missed.

#### AnnealedLeastCostMap

One way of trading off between optimality and complexity of the *LeastCostMap* heuristic is to apply a simulated annealing approach to decide whether to discard a higher cost partial map from the set in presence of a lower cost map. As the temperature of the process anneals, i.e. at the later iterations, the probability of keeping a non-minimal partial solution will decrease. Definitely this approach increases the computation and message complexity. However, this allows some of the non-minimal partial solutions to grow and possibly lead to a better complete solution.

#### RandomNeighbor

Another way of restricting the message complexity is to extend any partial map to a randomly chosen subset of kneighbors instead of expanding to all of them. Higher values of k increases the chance of getting the optimal solution. The *RandomNeighbor* heuristic with k = 1 did not produce results as good as LeastCostMap, although number of messages were reduced dramatically. Further investigation need to be done to determine a suitable value of k.

We compared the three heuristics in terms of quality of solution and message complexity. Because the optimal solution is hard to compute (NP-complete), we devised a relaxed version of the problem by relaxing the bandwidth constraints of the links. The solution to this relaxed problem can be computed in polynomial time. Any feasible solution for the actual problem will be feasible for the relaxed problem, thus the optimal solution to the relaxed problems gives a lower bound of the optimal cost. We computed the ratio of the cost of heuristic generated solutions to this lower bound cost. To assess the cost of executing the heuristics, we counted the total number of map-extension messages exchanged among the nodes. Because arrival of each map message invokes the processing algorithm on the receiving node, the total computational cost is proportional to the number of map messages.

Figure 3(a) shows that the heuristic derived solutions are fairly close to the lower bound of optimal solutions. We can observe that the *LeastCostMap* and the *AnnealedLeast-CostMap* heuristics perform well and get solutions that are very close to optimal solutions. We can see that the *RandomNeighbor* heuristic does not produce good solutions, because number of feasible ways to expand the partial maps narrows down very quickly here. In terms of cost of computation of the heuristics, we can observe in Figure 3(b) that number of map-extension messages to complete mapping of a single composition is much higher in the *AnnealedLeastCostMap* heuristic than the other two heuristics. These experiments show the *LeastCostMap* is the best among the three heuristics. We implemented the *LeastCostMap* heuristic in the map manager.

#### 3.1.3 Cost Metric

To devise a cost metric for choosing the best mapping among alternative feasible maps, we considered the following two factors - balancing the service workload among the servers and minimizing the uncertainty of using public network links where a dedicated link is available. The load-balance factor for a map (or a partial map) is com-



Figure 3. Comparing three heuristics

puted as an average of the server load-factors (ratio of used capacity to total capacity) for all the servers included in the map, and is always a number between 0 and 1. A map with lower load-balance factor spreads the components of a task on different servers rather than putting all of them into one, and chooses the under-utilized servers. In case two maps have almost same load-balance factor, (do not differ by more than 0.1 or 10%), then the one in which the number of hops (links connecting the processing components) assigned to dedicated links is higher is preferred. If that is also same, the map in which less number of hops are assigned to public network link is preferred.

### 3.2 Dynamic Re-allocation of Bi-modal Links

To effectively manage concurrent resource allocation requests and the variability in the flow-rate of the public network links, we need a mechanism to adapt with the changing network conditions. In this section we present the functional details of the dynamic scheduler component of the RMS that handles the dynamic reallocations.

The scheduler agent in each node is invoked periodically at regular intervals. The overall policy of the scheduler is to prioritize among competing tasks for use of the network links, based on their target data rate and offered price for the data processing service. When the platform accepts a task, the price of delivering each byte of data is specified in the task specification. This price is apportioned to each of the service components of the task, based on their processing requirements. Accordingly, a server executing a particular service will earn the apportioned price for each byte of data it processes. Besides executing the component services, servers may act as forwarding nodes for the stream processing tasks.

The links that carry the stream between two data processing servers can be of three different types -i) a direct dedicated link, ii) a multi-hop dedicated link through one or more forwarding nodes iii) an overlay link through the public network. A mapping of a task may contain any combination of these three types of links between the processing nodes. Among them, the direct dedicated links are the most preferred one, because they provide controlled and stable data rate. A multi-hop dedicated link provides similar control and stability, but it costs more because every forwarding node will charge the sender node for their forwarding task. This in turn reduces the revenue earned by the processing node for its work. Therefore, the price specification limits the number of indirections possible for a multi-hop link and a direct dedicated link is always preferred over a multi-hop link. The third possibility is having an overlay link through the public IP network. Because the sending node does not have any direct control over the packet routing in the public network, the flow rate is variable over such links. However, there is no additional perbyte cost for sending data through the overlay links. So, the nodes try to opportunistically use these links when dedicated links are overloaded or not available. Note that the server nodes are dedicated computers and thus, the allocated processing capacities to the task components do not vary over time. Hence there is no need to re-allocate the server resources after initial allocation.

At regular intervals when the scheduler is invoked, the procedure presented in Algorithm 2 is executed. The algorithm evaluates the fulfillment of processing rate requirement of each of the tasks being processed at the node, and re-allocates the available links of three types between this node and the node serving the next service component. While prioritizing among competing tasks, the scheduler tries to maximize the revenue earning of the server and prefers the tasks marked with higher price per unit of processing. On the other hand, the servers get penalized on the revenue, if they do not deliver the processed stream at the agreed upon rate. Therefore each server tries to fulfill the rate requirements of each task as much as possible. Thus, the task that requires more bandwidth to comply with its target gets higher preference. Hence the scheduler computes the priority of each task as a product of the apportioned price and the data rate required in next scheduling epoch.

To allocate the links, each node groups the tasks according to their next hop server. For each next hop group, highest priority tasks get allocation from the direct dedicated Algorithm 2 Link re-allocation algorithm

- 1: Invoked for each node *u* periodically
- 2: Group the tasks that are being processed in *u* by their next hop server *v*
- 3: for Each group v do
- 4: Compute the priority of each flow competing for a (u,v) link as -
- 5: priority = budget per byte of processed data \* bandwidth required to comply with the target rate
- 6: **if** any dedicated link (u,v) exists **then**
- 7: Assign the dedicated link to top priority flows until all capacity is used
- 8: end if
- 9: Collect all the unassigned flows
- 10: end for
- 11: **for** All the remaining flows **do**
- 12: **if** The budget permits k-hop (u,v) dedicated link, k > 1 **then**
- 13: Launch a probe search and reserve multi-hop dedicated path for the flow with maximum k hops
  14: Assign public network bandwidth for the flow
- temporarily
- 15: **else**
- 16: Assign public network bandwidth for the flow

#### 17: end if

18: end for

link, if such link exist and capacity permits. The next prior tasks are assigned multi-hop dedicated links. The maximum possible hops in such multi-hop links are restricted by the apportioned price for that task, because there is additional cost of forwarding at each hop and the processing node would not like to spend for forwarding cost beyond the amount of revenue it earns. The flows of the remaining tasks from all the groups are allocated bandwidth from the public overlay links.

# 4 Performance Evaluation and Discussion

## 4.1 Simulation Model

We constructed a simulation model of the distributed stream processing plaform according to the architecture and algorithms presented in Sections 2 and 3, respectively. The model was build on Java based simulation engine JiST [6].

Each of the servers in distributed locations are connected to the public Internet. Although each server has a certain uplink and downlink bandiwdth, the data rate over a connection that goes through the public network faces temporal variation. We use the statistics presented by Wallerich and Feldmann [15] to model the temporal variability of the end-to-end capacity of a path through the public network. From their data, the logarithm of the ratio of the observed transient flow rate to the mean flow rate over long period is almost a Normal distribution. In our simulations, all flows on the public network are perturbed every 10 milliseconds according to this model. With the allocated bandwidth as the mean rate and the standard deviation of the log-ratio set at 1, in 95% of the cases the observed bandwidth remains between one fourth  $(2^{-2\sigma})$  and four time  $(2^{2\sigma})$  of the allocated or mean bandwidth. Bandwidth of each last-mile connection (uplink and downlink) is randomly assigned between 1 Mbps and 2 Mbps.

In addition to the public network links, the servers are interconnected through dedicated links (which may be leased lines or privately installed links). For the dedicated network, we assume a preferential conectivity based network growth model similar to the one proposed by Barabasi et al [5]. The basic premise here is that when a server attempts to establish a dedicated link, it does so preferably with the most connected server. This eventually results in a power law degree distribution in the network. We assumed that server CPU capacity is proportional to the number of dedicated links it has. The variety of services that a server can host is also proportional to the node degree or capacity. The dedicated links have much higher bandwidth than the network links connecting a node to the public network. Their bandwidths were randomly assigned between 1 Mbps and 10 Mbps and the propagation delays were assumed to be between 1 and 10 milliseconds. The propagation delay of an end-to-end connection through the public network was much higher and assumed to be between 10 and 100 milliseconds.

Unless otherwise mentioned, we assumed the patform to have 100 server nodes and 99 dedicated links interconnecting them. There were 25 different types of services. As the service variety is proportional to the node degree, a node having d dedicated links was assumed to host 1 + ddifferent types of services (one added for public network link). Server CPU capacity was set such that it can execute k instances of each service concurrently, according to the mean data delivery rate. We set k = 2. For the task workload, each task is assumed to have 10 service components, randomly chosen from 25 different types of service. Mean data delivery rate was 1Mbps and total amount of data to be processed from the source was 100MB on average. Each data point on the results shown below is an average of 100 observations from different experiments on randomly generated networks with specified parameters. For each experiment, a synthetic workload trace containing 500 stream processing tasks were generated. The task arrival process is assumed to be Poisson, with the arrival rate varying accross the experiments. If not mentioned otherwise, the default arrival rate was 60 tasks per hour.

### 4.2 Benefits of Combining Opportunistic and Dedicated Resources

We performed several sets of experiments to evaluate the benefits of using bi-modal networks for stream processing tasks. In the experiments, we compare three possible settings -i) a network with the dedicated links only, ii) public network only, and iii) a network that combines both.

First argument in favor of a bi-modal network for stream processing is that combining the public network with dedicated links, the system achieves much higher work throughput at the same cost. To examine this, we fed similar workload traces under same arrival rates to two system set-ups, one with only dedicate link based networks and the other using the combination of dedicated links and public network. From Figure 4(b) we observe that for the same workload, if the platform uses dedicated links only, it needs more than 120 links to get 50% acceptance ratio, whereas the same acceptance ratio can be obtained with 50 dedicated links only, if the public network is utilized in conjunction. Similar evidence in Figure 4(a) shows that inclusion of the public network helps to achieve same overall system throughput at much lower number of dedicated link installations.

The next argument is that utilization of the privately deployed expensive dedicated resources such as servers and dedicated links is increased, if inexpensive public network is used in conjunction. From Figure 4(c) we observe that when a combination of dedicated links and the public network is used, the server utilization is higher than the sum of utilizations of cases using a single type of network links.

Figures 4(e) and 4(f) show another evidence of higher return on investment. In Figure 4(e), we observe that the utilization of dedicated links becomes consistently higher across a wide range of loading scenarios if the public network is used in combination. The lower utilization in case of a dedicated link only network results from the fact that the platform has rejected many task requests that would have been feasible by the augmentation of the public resources. Figure 4(f) shows the variation of utilization of the dedicated links with the number of dedicated links. We observe that the difference in utilization diminishes as the number of installed links increases. This is because when there is sufficient number of dedicated links to carry the required traffic of all the tasks, the public resources are not used at all, and the bi-modal system becomes equivalent to a dedicated link only system. In both cases, utilization of the links keeps decreasing when more and more links are added because the workload is held constant.

The discussion above highlighted the benefits of using public network towards improving the utilization of dedicated server and link resources (i.e., increases in return on investment). Next we investigate how the bi-modal network helps the stream processing platform to improve the compliance with the services contracts it has with indi-



Figure 4. Comparing bi-modal and uni-modal networks

vidual tasks. We measure the compliance of the stream processing platform as follows. Each task request specifies a time window T that is used to monitor the delivery rate. We measured the deviation from the required rate as  $\sum_{a}$  over all windows  $\frac{B-\hat{B}}{B}$ , where B is the desired rate and  $\hat{B}$  is the observed rate of delivery. In Figure 4(g), we observe that use of dedicated links brings the percent deviation down to between 10% and 20% from above 50%. In this case the number of installed dedicated links was just enough to make a spanning tree of the nodes, i.e. N-1links for N nodes. Note that deviation is counted on the accepted jobs only. So, even though for a dedicated link only network, the deviation is almost zero, we have seen that such network is unable to accept enough jobs to fully utilize the resources. In Figure 4(h), we observe that the deviation in the bi-modal system gets closer to zero as more and more dedicated links are added to the network. However, beyond certain number of links, (125 in this particular

experiment), the improvement is very marginal.

When we use a combination of dedicated and public links, it is expected that the completion time of each task will be slightly elongated compared to a system with only dedicated links, due to the variability in the public network. Nevertheless, using the combination contains the elongation to a small value, compared to the case where only public network is available. In Figure 4(i), we observe a 10 - 20% increase in the execution time in the bi-modal system, whereas execution time would be 200 - 300% more in case of a public network only system.

## 4.3 Necessity of Periodic Re-Scheduling

Another important question in managing the bi-modal stream processing platform is the importance of dynamic re-allocation of network links. The main intuition behind introducing dynamic re-allocation is that the flows that goes through the public network suffer from the variability



Figure 5. Effect of dynamic scheduling

and lag from the target rate, whereas the flows that uses dedicated links all-through, do not lag from the target at all. Dynamic scheduling introduces fairness across all the tasks. So if link assignment is done dynamically, it is expected to improve the utilization of the resources and increase the overall capacity of the system.

We fed the same workload to two system set-ups containing combinations of dedicated links and public network links. In one we disabled dynamic re-scheduling of links and let the tasks complete with the initial assignment of links and nodes. From Figures 5(a) we observe that overall system throughput increases with dynamic scheduling, as an indication of higher task acceptance ratio and higher utilization of the system resources. Figure 5(b) demonstrates that dynamic scheduling results in much higher utilization of the dedicated links. CPU utilization remains unchanged (not shown), because the dynamic re-allocation does not alter the node assignments. Another rationale behind reallocations is to increase fairness and improve compliance with the target delivery rate. Figure 5(c) shows that irrespective of workload, the dynamic scheduling decreases the deviation from the specified target, having the same number of dedicated links and same public network bandwidth.

# 5 Related Work

Although there is a vast body of literature on resource management in cluster, Grid or peer-to-peer hosting platforms, there have been relatively a very few works that proposes combined use of dedicated and public resources. In [10], Kenyon et al. provided arguments based on mathematical analysis, that commercially valuable quality assured services can be generated from harvested public computing resources, if small amount of dedicated computers can be augmented with them. With simple models for available periods of harvested cycles, their work have measured the amount of dedicated resources necessary to achieve some stochastic quality assurance from the platform. However, they did not study how a bi-modal platform would perform in the presence of clients with different service level agreements and how to engineer the scheduling policies to maximize the adherence to these agreements.

Recently, in [7], Das et al. have proposed the use of dedicated streaming servers along with BitTorrent, to provide streaming services with commercially valuable quality assurances while maintaining the self scaling property of Bit-Torrent platform. With analytical models of BitTorrent and dedicated content servers they have demonstrated how guaranteed download time can be achieved through augmentation of these platforms. However, their proposal does not include actual protocols that can be used to achieve these performance improvements.

Architectures and resource management schemes for distributed stream processing platforms have been studied by many research groups from distributed databases, sensor networks, and multimedia streaming. In database and sensor network research, the major focus was placing the query operators to nodes inside the network that carries the data stream from source to the viewer [13]. In multimedia streaming problems, similar requirements arise when we need to perform a series of on-line operations such as transcoding or embedding on one or more multimedia streams and these services are provided by servers in distributed locations. In both cases, the main problem is to allocate the node resources where certain processing need to be performed along with the network bandwidths that will carry the data stream through these nodes.

Finding the optimal solution to this resource allocation problem is inherently complex. Several heuristics have been proposed in the literature to obtain near-optimal solutions. Recursive partitioning of the network of computing nodes have been proposed in [11] and [14] to map the stream processing operators on a hierarchy of node-groups. They have demonstrated that such distributed allocation of resources for the query operators provides better response time and better tolerance to network perturbations compared to planning the mapping at a centralized location.

In [16] and [8], the service requirements for multi-step processing of multimedia streams, defined in terms of service composition graphs have been mapped to an overlay network of servers after pruning the whole resource network into a subset of compatible resources. The mapping is performed subject to some end-to-end quality constraints, but the CPU requirements for each individual service component is not considered. Liang and Nahrstedt in [12] have proposed solutions to the mapping problem where both node capacity requirement and bandwidth requirements are fulfilled. However, one of the assumptions made by Liang and Nahrstedt was that the optimization algorithm was executed in a single node and complete state of the resource network is available to that node before execution. In a large scale dynamic network this assumption is hard to realize. If we assume that each node in the resource network is aware of the state of its immediate neighborhood only, we need to compute the solution using a distributed algorithm such as ours.

In all of the abovementioned works, the operator nodes are assumed to interconnected through an application dependent overlay network using the Internet as underlay. In [9], Gu and Nahrstedt presented a service overlay network for multimedia stream processing, where they have shown that dynamic re-allocation of the operator nodes provides better compliance with the service contracts in terms of service availability and response time. However, none of the works have proposed the use of dedicated links in conjunction with IP overlay network for improving adherence to the service contracts.

### 6 Conclusion

In this paper, we investigated the resource management problem with regard to data stream processing tasks. In particular, we examined how a hybrid platform made up of dedicated server resources and bi-modal network resources (dedicated plus public) can be used for this class of applications. From the simulation based investigations, we were able make several interesting observations. First, bimodal networks can improve dedicated resource utilization (server plus dedicated network links). This means higher return on investment can be obtained by engaging the bimodal network. Second, the overall system is able to admit and process tasks at a higher rate compared to system configurations that do not leverage a bi-modal network. Because the public network is engaged at zero or very low cost, this improvement in throughput can be result in significant economic gain for institutions that perform data stream processing workloads. Third, the engagement of bi-modal network comes at a slight overhead that adds low delays in stream processing tasks. Compared to publiconly networks the delays provided by the bi-modal network is almost negligible. Fourth, dynamic rescheduling is essential to cope with varying network conditions - particularly in the public network. The dynamic rescheduling algorithm switches the flows according to the recomputed priority values to achieve the best service level compliances.

In summary, our study highlights the benefits of the bimodal architecture for compute- and network-intensive applications. Moreover, it provides simple distributed algorithms that allows the effective utilization of such a platform for data stream processing applications. Deploying the distributed resource management framework in an actual prototype for data stream mapping is a possible future work.

### References

- S. Asaduzzaman. Managing Opportunistic and Dedicated Resources in a Bi-modal Service Deployment Architecture. PhD thesis, School of Computer Science, McGill University, Oct. 2007.
- [2] S. Asaduzzaman and M. Maheswaran. Utilizing Unreliable Public Resources for Higher Profit and Better SLA Compliance in Computing Utilities. *Journal of Parallel and Distributed Computing*, 66(6):796–806, 2006.
- [3] S. Asaduzzaman and M. Maheswaran. Strategies to Create Platforms for Differentiated Services from Dedicated and Opportunistic Resources. *Journal of Parallel and Distributed Computing*, 67(10):1119–1134, 2007.
- [4] S. Asaduzzaman and M. Maheswaran. Towards a decentralized algorithm for mapping network and computational resources for distributed data-flow computations. In 21st Annual International Symposium on High Performance Computing Systems and Applications, page 30, May 2007.
- [5] A. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.
- [6] R. Barr, Z. J. Haas, and R. van Renesse. JiST: An efficient approach to simulation using virtual machines. *Software: Practice and Experience*, 35(6):539–576, 2005.
- [7] S. Das, S. Tewari, and L. Kleinrock. The Case for Servers in a Peer-to-Peer World. In *Proceedings of IEEE International Conference on Communications (ICC '06)*, pages 331–336, Jun. 2006.
- [8] X. Gu and K. Nahrstedt. Distributed multimedia service composition with statistical QoS assurances. *IEEE Trans. Multimedia*, 8(1):141–151, 2006.
- [9] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. Qosassured service composition in managed service overlay networks. In 23rd International Conference on Distributed Computing Systems, pages 194–203, May 2003.
- [10] C. Kenyon and G. Cheliotis. Creating Services with Hard Guarantees from Cycle Harvesting Resources. In 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), May 2003.
- [11] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource aware distributed stream management using dynamic overlays. In *Proc. 25th IEEE ICDCS*, pages 783–792, Jun. 2005.
- [12] J. Liang and K. Nahrstedt. Service composition for generic service graphs. *Multimedia Systems*, 11(6):568–581, 2006.

- [13] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings* of the 22nd International Conference on Data Engineering, ICDE 2006, page 49, Apr. 2006.
- [14] S. Seshadri, V. Kumar, B. F. Cooper, and L. Liu. Optimizing Multiple Distributed Stream Queries Using Hierarchical Network Partitions. In *Proceedings of 21th International Parallel and Distributed Processing Symposium* (*IPDPS 2007*), pages 1–10, Mar. 2007.
- [15] J. Wallerich and A. Feldmann. Capturing the variability of internet flows across time. In 25th IEEE International Conference on Computer Communications (INFOCOM-2006), Apr. 2006.
- [16] M. Wang, B. Li, and Z. Li. sFlow: Towards resourceefficient and agile service federation in service overlay networks. In *Proc. 24th IEEE ICDCS*, pages 628–635, Mar. 2004.