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

Shah Asaduzzaman
SITE, University of Ottawa
Ottawa, ON, K1N 6N5, Canada
Email: asad@site.uottawa.ca

Muthucumaru Maheswaran
SOCS, McGill University
Montreal, QC H3A 2A7, Canada
Email: maheswar@cs.mcgill.ca

Abstract

This paper presents a case for exploiting the synergy of dedicated and opportunistic network resources in a distributed hosting platform for data stream processing applications. Our previous studies have demonstrated the benefits of combining dedicated reliable resources with opportunistic resources in case of high-throughput computing applications, where timely allocation of the processing units is the primary concern. Since distributed stream processing applications demand large volume of data transmission between the processing sites at a consistent rate, adequate control over the network resources is important here to assure a steady flow of processing. In this paper, we propose a system model for the hybrid hosting platform where stream processing servers installed at distributed sites are interconnected with a combination of dedicated links and public Internet. Decentralized algorithms have been developed for allocation of the two classes of network resources among the competing tasks with an objective towards higher task throughput and better utilization of expensive dedicated resources. Results from extensive simulation study show that with proper management, systems exploiting the synergy of dedicated and opportunistic resources yield considerably higher task throughput and thus, higher return on investment over the systems solely using 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 [1, 2]. Depending on the application, the stream processing tasks can have complex topologies with multiple sources or multiple sinks. Examples of stream processing tasks are found 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 analysis of network monitoring data streams for malicious activity detection, and (iv) function computation over data feeds obtained from sensor networks.

One of the salient characteristics of this class of applications is the demanding compute and network resource requirements [3]. Huge volume of data generated at a high rate need to be processed within real-time constraints. Moreover, various operations on these data streams are provided by different servers at distributed geographic locations [4]. All these factors demand a scalable and adaptive architecture for distributed stream processing platform, where fine-grained control over processing and network resources is possible.

Earlier works on stream processing engines [5] resorted to centralized single-server or server-cluster based solutions where tighter control over available resources are possible. With possibility of different processing services being provided by different providers, need for distributed stream processing platform arose. Several architectures have been proposed to support such distributed processing of streams [3, 6, 4, 7]. Due to the stringent rate-requirement for processing and transmission of data, most researchers have assumed a central resource controller that can gather the availability status of all resources and map the requested tasks on them. However, with advent of diverse range of stream processing services, it is important to allow autonomous providers of services to collaborate and share their resources. Thus it is important to develop distributed resource allocation schemes, where control is available over local resources only.

While it is feasible to have dedicated server resources and precisely allocate them for processing tasks, dedicated networks over wide-area installations remain costly. It is possible to propagate the data streams through the the distributed servers using the public Internet. However, the lack of adequate control over end-to-end bandwidth in current Internet and the stringent rate requirement of the stream processing applications demand some dedicated network resources. In fact, recent advances in optical network technologies such as user-controlled light path [8] open the possibility of on-demand provisioning of end-to-end optical links with total control of the bandwidth available to the user application.

In this paper, we explore a novel approach where a combination of dedicated links and public network to interconnect the servers. The main focus of this paper is to explore how such a hybrid (denoted *bi-modal* in this paper) network can be best used for data stream processing tasks. The hypothesis that drives this work is that the combination has a synergistic effect that allows better utilization of the dedicated resources, and yields higher return on investment. We devised distributed algorithms for allocation of these hybrid resources to demonstrate the viability of this synergy hypothesis.

This paper extends some of our previous work [9, 10] on bi-modal compute platforms where dedicated compute-clusters were augmented with opportunistically harvested processing elements to increase work throughput and 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 communication-intensive applications.

In Section 2 we present the system model for the data stream processing and the associated resource allocation problem. Section 3 discusses the algorithms devised for managing the resources towards global optimization of throughput and resource utilization. Section 4 examines the results from the extensive simulation studies we carried out to evaluate the algorithms.

2 System Model and Assumptions

In a *stream processing task*, the data stream originating from a *data-source* node, progresses through several steps of processing, termed as *service components* (or *service* in short), before being delivered to the *data-delivery* node. Although, in very general terms, the data-flow topology could be arbitrary graphs, in this paper, we restrict our study within linear path topology only.

The distributed stream processing platform consists of several autonomous server nodes that serve the service components. A single server may serve multiple services and a service may be available at multiple servers. Several pairs of servers establish dedicated point-to-point links between them to have the flow of the data streams at a controlled rate. Each server is also connected to the public Internet and end-to-end TCP connection can be established between any pair of servers. However, end-to-end bandwidth of the TCP connections cannot be allocated and the flow rate cannot be controlled.

The platform is modeled as an asynchronous message passing distributed system, where there is no centralized controller to coordinate the resources. The servers have knowledge of and can precisely allocate the local resources only, i.e. the processing capacity and the bandwidth of outgoing links. However, the servers comply with the global protocol and respond to a predefined set of messages in a predefined way. The objective of the global protocol is to ensure adequate resources for each individual task for its seamless progress, and to maximize the global work throughput. Other factors such as balancing the load among different servers and maximizing the utilization of dedicated resources are also considered. Design and evaluation of the protocol constitute the remaining sections of the paper.

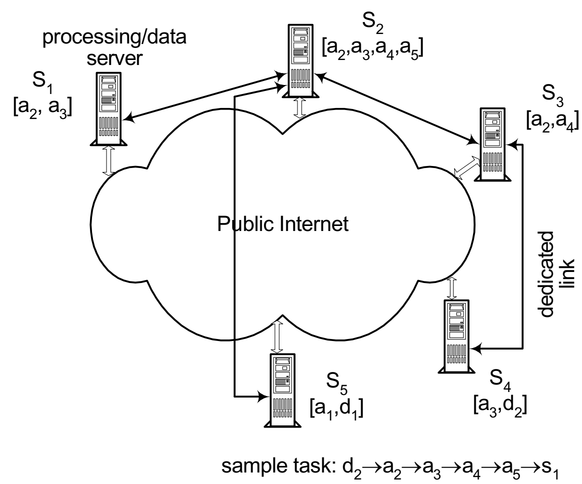


Figure 1. Stream processing platform

Figure 1 illustrates a scenario of a stream processing platform containing five servers. 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 S_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.

For convenience, the resource allocation process is divided into two phases. First, individual tasks with multiple service components are mapped on the processing servers fulfilling the processing and transport capacity requirements. A cost function is used to select the best among multiple feasible maps. The second phase re-allocates the link bandwidths among competing tasks, after the tasks start execution based on the initial allocation. This is necessary because of the variability of data rate in the end-to-end TCP connections on public Internet. Both the re-allocation phases and initial allocation are driven by the same global optimization goal, namely maximization of global throughput and resource utilization, subject to fulfillment of individual task requirements.

The specification of a stream processing task includes the ordered sequence of service components, the data source node, the data delivery node and the desired rate of data delivery. We assume a rate based model [3, 7] to specify resource requirement for each service component. For any service, both the output data rate and the CPU requirement are proportional to the input data rate, and are specified by two factors – the *bandwidth shrinkage factor* and the *CPU usage factor*, respectively. We also assume a rate based pricing for the services. The task specification includes a price per byte of data delivered. This is directly translated to apportioned prices for each of the service components, using the above two factors. The task specification is a *service level agreement* (SLA) between the user and the platform.

3 Decentralized Management of Server and Network Resources

A resource management engine (denoted as RMS agent) runs in each server that implements the protocols for coordinated allocation of resources. It has two modules – a map manager and a dynamic scheduler, to perform the two phases of resource allocation described before. This section describes the functions of these two modules in details.

A user of the platform uses one of the server nodes as a portal to launch its stream processing task. The portal node then engages the map manager 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 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 selected.

A reservation probe is then sent to the data-delivery node along the path found in the selected map. The RMS agent at each server node along the path attempts to allocate the server and link resources prescribed by the map. Because the mapping process for multiple tasks may be ongoing concurrently, it is possible that the required resource is no longer available. In such case the allocation fails, the probe is rolled back and the next feasible map is probed by the data-source node. Once a successful probe reaches the data-delivery node at the other end, a confirmation is sent back to the data-source node to begin the streaming. The message flow of mapping and reservation is illustrated in Figure 2.

The dynamic scheduler in each server node periodically re-allocates the locally available link resources among the competing tasks that are using that server node. The re-allocation process is illustrated in Figure 3. The re-allocation is done with two objectives – improving the compliance to the SLA defined data-delivery rate and maximizing the global processing throughput. Note that only link resources are re-allocated while keeping the allocation of server resources unmodified. This follows from the assumption that servers are dedicated and their processing rates do not vary over time.

3.1 Distributed Algorithm for Mapping

The problem of mapping a stream processing task specification on arbitrary network of server nodes subject to processing capacity and bandwidth constraints is an NP-Complete problem [11]. Detailed analysis of the problem and algorithms to solve it in both centralized and distributed manner can be found in [12, 11]. Algorithm for a similar mapping problem was also discussed in [4]. However, use of bi-modal network links was not considered before. Here, we adopt the distributed mapping algorithm presented in [11] with modifications to accommodate bi-modal network links.

The distributed mapping of the task specification performed by gradually expanding the maps to neighbors in the server network. The portal server initiates the algorithm by generating the initial map message. The *ProcessMap* algorithm described in Algorithm 1 is executed by the map manager at each server node on receiving a map message.

The algorithm first extends the received partial map by mapping next few service components on itself as long as the service is available and processing capacity permits (line 6–8). Each possible extension is then sent to neighboring nodes subject to availability of network bandwidth (line 11–22). Note that it is possible to extend the map to the neighbors without having any service mapped on the current server. This allows multi-hop connection between nodes processing consecutive services. This is beneficial in cases where there is no direct dedicated link between two server nodes.

In case of links through the public network, such multi-hops are unnecessary, because overlay link can be established between any pair of nodes. This case is handled in lines 16–22. Note that end-to-end bandwidth cannot be allocated in case of public network links,

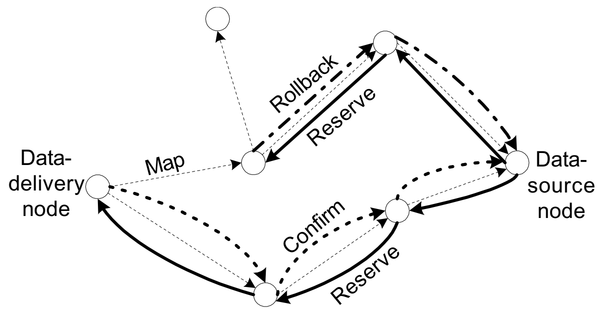


Figure 2. Mapping, reservation and rollback

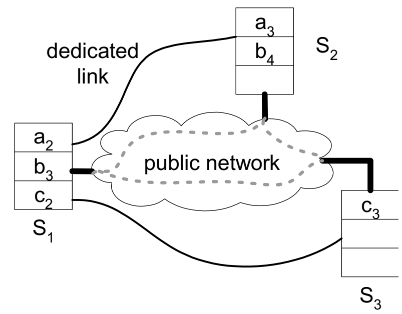


Figure 3. Dynamic scheduling of link resources done by server S_1 on three competing task segments a_2-a_3, b_3-b_4, c_2-c_3

only uplink bandwidth can be controlled. In case of extending through a public network link, only the nodes that provide the service required in the next hop is chosen (line 17). We assume that an underlying gossip like algorithm disseminates the presence of services in each server across the network. Thus, for each service type, each node has the knowledge of (possibly a subset of) the nodes that hosts that service. Incorrectness of this information does not cause any inconsistent mapping, only some feasible maps are missed.

Cyclic mapping is allowed in the extension in lines 12–14. Because $x = 0$ is allowed, it is possible that the map grows to an infinite length. In practice, this is avoided by limiting the growth of the multi-hop mapping using a budget factor. Based on the price-per-byte-processed quoted in the SLA, the allocated revenue for processing of the $j - th$ service is limited. When the output of the j-th service is sent to the server providing $(j + 1)$th service using a dedicated link, host of the j-th service needs to pay and thus loses revenue. The cost of transmission grows as more dedicated links are used in a multi-hop link to send the same data. Thus the number of hops in such multi-hop links are limited by the revenue budgeted for the service and cost of each hop of dedicated connection.

Because the algorithm enumerates all feasible maps, it generates an exponential number of messages. Some simple heuristics can limit the complexity without sacrificing much of optimality. We have used a simple heuristic called *LeastCostMap* where each node remembers the lowest cost map it has observed so far for each possible prefix length (number of components already mapped), and it does not extend a map with higher cost for the same prefix length. Evaluation of performance of the heuristic compared to other possible heuristics can be found in [12].

To devise an appropriate cost metric for choosing the best 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. The load-balance factor for a map (or a partial map) is computed 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 with least number of hops through public network is preferred.

3.2 Algorithm for Dynamic Link Re-allocation

The dynamic link scheduler in each node is invoked periodically at regular intervals. Based on current evaluation of locally observed performance, the scheduler reallocates the locally available link resources among the competing tasks that are using this node. The overall policy of the scheduler is to prioritize the tasks for use of the network links, based on their deviation from target data rate and the price they would pay for the data processing service.

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

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

Input: 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
if u is the data-source node and all the services except the data source is mapped in m **then**
  m is a feasible map
else
  for $x = 0$ to $|T| - j - 1$ **do**
    if service $j + x$ is provided by node u and node capacity permits the required processing rate **then**
      m_x = map found by extending next x services in T on u
    else
      break
    end if
    for each node v such that there is a dedicated link (u, v) **do**
      if (available bandwidth in (u, v) link $\geq$ the bandwidth need for the service hop $(j+x, j+x+1)$) and budget allows the extension of m_x to v **then**
        Send m_x to v
      end if
    end for
    if $x > 0$ **then**
      for each node v such that v provide the service $j + x + 1$ **do**
        if available uplink bandwidth to the Internet $\geq$ bandwidth need for service hop $(j + x, j + x + 1)$ **then**
          Send m_x to v
        end if
      end for
    end if
  end for
end if

they provide controlled and stable data rate. A multi-hop dedicated link provides similar control and stability, but it costs more (Section 3.1). The third possibility is having an overlay link through the public network. The flow rate is variable over such links, but there is no additional cost for sending data through them. So, the nodes try to opportunistically use these links when dedicated links are overloaded or not available.

Algorithm 2 is executed when the scheduler is invoked. For allocation of the links, tasks are grouped according to their next hop server node (line 2). While prioritizing among competing tasks for each group (lines 4-5), 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 server tries to fulfill the rate requirement of each task, because it gets penalized otherwise. 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.

For each next hop group, highest priority tasks get allocation from the direct dedicated link, if such link exist and capacity permits (lines 6-8). The next prior tasks are assigned multi-hop dedicated links (lines 12-14). The maximum possible hops in such multi-hop links are restricted by the apportioned price for that service according to the task specification. The remaining tasks from all the groups are allocated bandwidth from the public overlay links (lines 15-17).

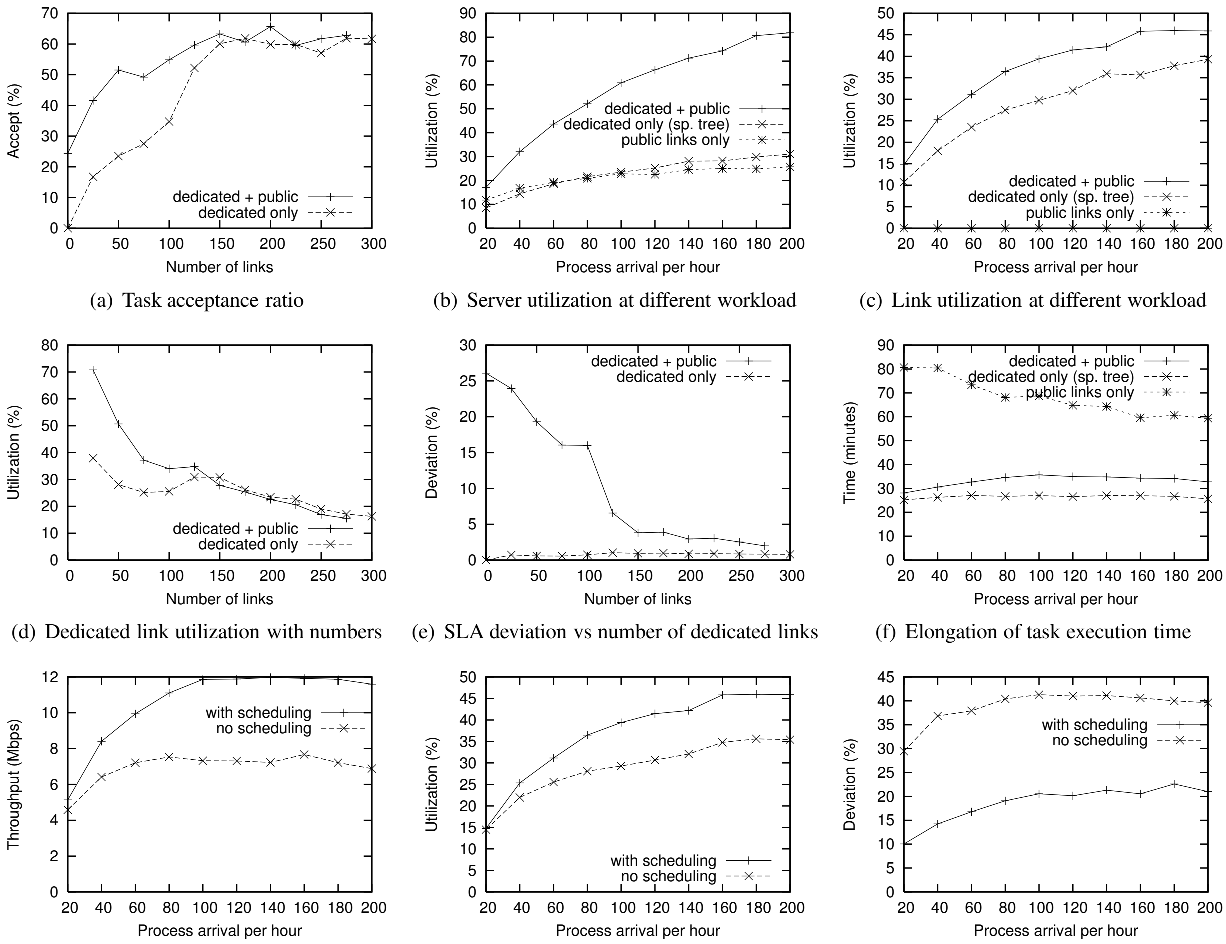
Algorithm 2 Link re-allocation algorithm

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 $\leftarrow$ 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**

4 Performance Evaluation and Discussion

4.1 Simulation Model

We constructed a simulation model of the proposed distributed stream processing platform using Java based discrete event simulator JiST [13]. Each server in the platform is connected to the Internet using last mile bandwidth between 1 Mbps and 2 Mbps, randomly



(a) Task acceptance ratio (b) Server utilization at different workload (c) Link utilization at different workload

(d) Dedicated link utilization with numbers (e) SLA deviation vs number of dedicated links (f) Elongation of task execution time

(g) Throughput (h) Utilization of dedicated links (i) Deviation from target rate

Figure 4. (a-f) Bi-modal vs uni-modal networks. (g-i) Effect of periodic re-scheduling

assigned. To model the variability of data rate on end-to-end Internet paths, we used the statistics presented by Wallerich and Feldmann [14]. From their data collected from packet level traces from core routers of two major ISPs over 24 hours, 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 ms 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.

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 connectivity based network growth model similar to the one proposed by Barabasi et al [15]. 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 ms. The propagation delay of an end-to-end connection through the public network was much higher and assumed to be between 10 and 100 ms.

Unless otherwise mentioned, we assumed the platform 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 $1 + d$ different 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 across the experiments. If not mentioned otherwise, the default arrival rate was 60 tasks per hour.

4.2 Benefits of Combining Opportunistic and Dedicated Resources

To evaluate the benefits of using bi-modal networks for stream processing, we run several simulation experiments in 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(a) 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. As a result, the bi-modal system yields much higher work throughput for same number of dedicated links.

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(b) we observe that when a combination of dedicated links and the public network is used, the server utilization is much higher than the utilization when only one type of link is used. The synergy of bi-modal links is evident here, because, when sufficiently loaded, the server utilization of bi-modal system is higher than the sum of utilizations in the two other cases.

Figures 4(c) and 4(d) show another evidence of higher return on investment. In Figure 4(c), 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(d) 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.

Next, we investigate how the bi-modal network helps the stream processing platform to keep the compliance with the services contracts it has with individual tasks. We measure the compliance 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 $\sum_{\text{over all windows}} \frac{B-\hat{B}}{B}$, where B is the desired rate and $\hat{B}$ is the observed rate of delivery. In Figure 4(e), 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. 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.

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(f), we observe a $10 - 20\%$ increase in the execution time in the bi-modal system, whereas the elongation is $200 - 300\%$ in a public network only system.

4.3 Necessity of Periodic Re-Scheduling

Another important question in managing the bi-modal 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 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 4(g) 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 4(h) 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 re-allocations is to increase fairness and improve compliance with the target delivery rate. Figure 4(i) 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 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, bi-modal networks can improve dedicated resource utilization (server plus dedicated network links). This means higher return on investment can be obtained by engaging the bi-modal 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 small delays in stream processing tasks. Compared to public-only 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 bi-modal architecture for data-intensive stream processing applications and provides simple distributed algorithms for effective utilization of the resources.

References

[1] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. In *ICDE*, page 49, Apr. 2006.

[2] S. Seshadri, V. Kumar, B. F. Cooper, and L. Liu. Optimizing Multiple Distributed Stream Queries Using Hierarchical Network Partitions. In *IPDPS*, pages 1–10, Mar. 2007.

[3] Y. Drougas and V. Kalogeraki. RASC: Dynamic Rate Allocation for Distributed Stream Processing Application. In *IPDPS*, Mar. 2007.

[4] J. Liang and K. Nahrstedt. Service composition for generic service graphs. *Multimedia Systems*, 11(6):568–581, 2006.

[5] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *CIDR-2003*, Jan. 2003.

[6] 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.

[7] T. Kichkaylo and V. Karamcheti. Optimal Resource-Aware Deployment Planning for Component-based Distributed Applications. In *HPDC*, pages 150–159, Jul. 2004.

[8] W. Golab and R. Boutaba. Path Selection in User-controlled Circuit-switched Optical Networks. *Optical Switching and Networking*, 5(2-3):123–138, Jun. 2008.

[9] 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.

[10] 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.

[11] S. Asaduzzaman and M. Maheswaran. Towards a decentralized algorithm for mapping network and computational resources for distributed data-flow computations. In *21st IEEE HPCS*, page 30, May 2007.

[12] S. Asaduzzaman. *Managing Opportunistic and Dedicated Resources in a Bi-modal Service Deployment Architecture*. PhD thesis, School of Computer Science, McGill University, Jan. 2008.

[13] 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.

[14] J. Wallerich and A. Feldmann. Capturing the variability of internet flows across time. In *INFOCOM*, Apr. 2006.

[15] A. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.