

Distributed Resource Management and Admission Control of Stream Processing Systems with Max Utility

Cathy H. Xia
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
cathyx@us.ibm.com

Don Towsley * Chun Zhang *†
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
{towsley,czhang}@cs.umass.edu

Abstract

A fundamental problem in a large scale decentralized stream processing system is how to best utilize the available resources and admission control the bursty and high volume input streams so as to optimize overall system performance. We consider a distributed stream processing system consisting of a network of servers with heterogeneous capabilities that collectively provide processing services to multiple data streams. Our goal is to design a joint source admission control, data routing, and resource allocation mechanism that maximizes the overall system utility. Here resources include both link bandwidths and processor resources. The problem is formulated as a utility optimization problem. We describe an extended graph representation that unifies both types of resources seamlessly and present a novel scheme that transforms the admission control problem to a routing problem by introducing dummy nodes at sources. We then present a distributed gradient-based algorithm that iteratively updates the local resource allocation based on link data rates. We show that our algorithm guarantees optimality and demonstrate its performance through simulation.

Keywords: Stream Processing, Distributed Algorithms, Multicommodity Flow Model, Gradient Methods

1 Introduction

Enable by recent advances in computer technology and wireless communications, a new set of stream processing applications flourish in a number of fields ranging from environmental monitoring, financial analysis, and system diagnosis to surveillance/security and industrial control. At

*This work was supported in part by the National Science Foundation under grant EEC-0313747.

†Chun Zhang is now at IBM T.J. Watson Research Center, Hawthorne, NY; Email: czhang1@us.ibm.com. This work was done while he was a student at UMass, Amherst.

the core of these applications is a stream processing engine that performs resource allocation and management to support continuous tracking of queries over collections of physically-distributed and rapidly updating data streams. Distributed stream processing architecture has emerged as an appealing solution in response to these applications. In recent years, a number of stream processing systems have been developed, see, for example, Borealis [1], Medusa [9], GATES [8], and System S [12].

In most of today's distributed stream processing systems, massive numbers of real-time streams enter the system through a subset of the processing nodes. The processing nodes may be co-located within a single cluster, or geographically distributed over wide areas, hence both network and processor resources are constrained. The streams have diverse processing and transmission requirements, the results of which are directed to sinks. In order to carry out the processing, the limited computational resource of a node needs to be divided among the possibly multiple streams passing through the node either using time-sharing of the processor, or a parallel processing mechanism. The rates at which data arrive can be bursty and unpredictable, which can create a load that exceeds the system capacity during times of stress. Even when the system is not stressed, in the absence of any type of control, the initiation of these streams is likely to cause congestion and collisions as they traverse interfering paths from the plurality of sources to the sinks. Given that each node has only local knowledge of the network condition, it is difficult to determine the best control mechanism at each node in isolation. The system needs to coordinate processing, communication, buffering, and the input/output of neighboring nodes so that the overall system performance is optimized. The design of such a joint admission control, data routing, and resource allocation mechanism is therefore of great importance.

Previous work on resource management for distributed stream processing systems has focused on either heuristics

for avoiding overload or simple schemes for load-shedding (e.g. [15, 3, 7]). To the best of our knowledge, the joint problem of dynamic admission control and distributed resource management that maximizes overall system utility has not yet been fully studied.

In this paper, we present a distributed algorithm for the optimal joint allocation of *processing* and *communication* resources in a generic stream processing system. To make the problem concrete, we consider a stream processing network consisting of many servers, collectively providing processing services for multiple data streams. Each stream is required to complete a series of operations on various servers. The stream data rate may change after each operation. For example, a filtering operation may shrink the stream size, while a decryption operation may expand the stream size. Thus our corresponding flow network differs from the conventional flow network since flow conservation, in the classical sense, no longer holds. We assume all servers have finite computing power and all communication links have finite available bandwidth. We further assume that the performance of a stream is captured by an increasing concave utility function that takes the stream data rate as its argument. Our goal is to design a joint source admission control, data routing, and resource allocation mechanism so as to maximize the sum of utilities.

Our approach is to map the problem into a multicommodity flow network and then address the admission control and resource allocation simultaneously for general utility functions. Such approaches have been used to solve various routing problems in the areas of networking [10, 13]. Our work differs from these efforts in multiple aspects.

First, we generalize the multicommodity model [4] to the stream processing setting which allows flow shrinkage and expansion. Multicommodity flow problems have been studied extensively in the context of conventional flow networks. Readers are referred to [4, 2] for the solution techniques and the related literature. Traditional multicommodity flow networks require flow conservation, which no longer holds with flow shrinkage/expansion.

The traditional wired/wireless network optimization formulation [10, 13] often assumes constraints on link-level capacities. In our problem, in addition to the link bandwidth constraints, we also have processing power constraints for each server. We present an extended graph representation of the problem that unifies the two different types of resources and the resulting network only has resource constraints on the nodes.

We present a novel scheme that maps the admission control problem into a routing problem, using the so-called dummy nodes to accommodate the (initially unknown) source input rates. This also enhances our earlier work [6], which handles linear utility functions and assumes that the desired source input rates are known.

We then present a distributed algorithm to solve the resulting routing problem and show that our algorithm converges to the optimal solution. The algorithm can be considered as a generalization of [10], which is a gradient-based algorithm that iteratively updates the local resource allocation based on link data rates.

The rest of the paper is organized as follows: Sections 2-3) present the model and transform the original problem into an equivalent but more tractable formulation. Section 4 presents a distributed algorithm that solves the problem. The performance of the proposed algorithm is then demonstrated through numerical examples in Section 6. Finally, concluding remarks are presented in Section 7.

2 The Stream Processing Model and Problem Formulation

The System Model: We consider a distributed stream processing system consisting of a network of cooperating servers. We model the underlying physical network as a capacitated directed graph $G_0 = (\mathcal{N}_0, \mathcal{E}_0)$ where \mathcal{N}_0 denotes the set of processing nodes, sensors (data sources), and sinks, and \mathcal{E}_0 denotes the connectivity between the various nodes. Associated with each node is a processing constraint, $C_u, u \in \mathcal{N}_0$ and with each link a communication bandwidth $B_{i,k}, (i, k) \in \mathcal{E}_0$. We assume that sources can process data whereas sinks cannot; they only receive data. Hence we find it useful to separate them into sets \mathcal{P} and \mathcal{J} where $\mathcal{N}_0 = \mathcal{P} \cup \mathcal{J}$. Graph G_0 can be arbitrary.

Commodities: Corresponding to the multiple concurrent applications or services supported by the system, the system needs to process various streams and produce multiple types of eventual information or query products for different end-users. We assume that queries are processed independently of each other, although they may share some common computing/communication resources. We refer to these different types of eventual processed information as *commodities*. We assume there are J different types of commodities, $J = |\mathcal{J}|$, each associated with a unique source node, $s_j \in \mathcal{P}$ and a unique sink node $j \in \mathcal{J}$. We assume that source s_j can generate data up to a maximum rate λ_j .

Each commodity is required to complete a series of operations or tasks on various servers before reaching the corresponding sink. A task may be assigned to multiple servers, and tasks belonging to different commodities may be assigned to the same server. Effective placement of various tasks onto the physical network itself is an interesting problem and useful techniques can be referred to [14]. Here, we assume the task to server assignment is given. For simplicity, we assume a server is assigned to process at most one task for each commodity.

Based on the task to server assignment, the tasks of each commodity stream form a *directed acyclic graph*(DAG)

$G_j = (\mathcal{N}_j, \mathcal{E}_j)$ where $\mathcal{N}_j \subseteq \mathcal{N}_0$ and $\mathcal{E}_j \subseteq \mathcal{E}_0$, $j \in \mathcal{J}$.

Generic Graph representation: We can now represent the problem using a generic (directed) graph $G = (\mathcal{N}, \mathcal{E})$ where $G = \cup_{j \in \mathcal{J}} G_j$. Here $\mathcal{N} \subseteq \mathcal{N}_0$, which consists of source/processing nodes and sink nodes, and $\mathcal{E} \subseteq \mathcal{E}_0$. An edge $(i, k) \in \mathcal{E}$ for server i indicates that a task resides on node k that can handle data output from node i for some commodity. Graph G is assumed to be connected. Note that G itself may not be acyclic, however, the subgraphs corresponding to individual streams are DAGs.

Consider, for example, a system with 8 servers and 2 streams. Stream S1 requires the sequential processing of Tasks A, B, C, and D, and Stream S2 requires the sequence of Tasks G, E, F, and H. Suppose the tasks are assigned such that $\mathcal{T}_1 = \{A\}$, $\mathcal{T}_2 = \{B\}$, $\mathcal{T}_3 = \{B, E\}$, $\mathcal{T}_4 = \{C\}$, $\mathcal{T}_5 = \{C, F\}$, $\mathcal{T}_6 = \{D\}$, $\mathcal{T}_7 = \{G\}$, $\mathcal{T}_8 = \{H\}$, where \mathcal{T}_i denotes the set of tasks that are assigned to server i . Then the directed acyclic sub-graph of the physical network is shown in Figure 1, where the sub-graph composed of solid links corresponds to stream S1 and the sub-graph composed of dashed links corresponds to stream S2. It is easily verified that the sub-graphs corresponding to individual streams are directed acyclic graphs (DAG).

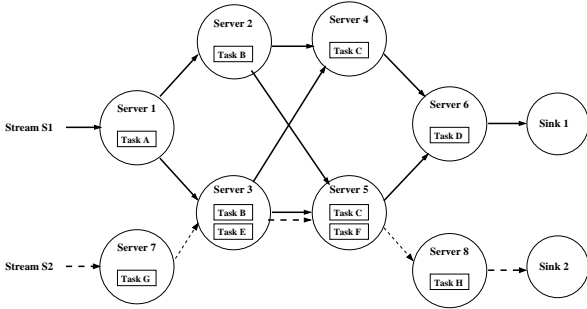


Figure 1. Physical Server Graph

We assume that it takes computing power $c_{u,v}^{(j)}$ for node $u \in \mathcal{N}$ to process one unit of commodity j flow for downstream node v with $(u, v) \in \mathcal{E}$. Each unit of commodity j input produces $\beta_{u,v}^{(j)} (> 0)$ units of output after processing. This parameter β only depends on the task being executed for its corresponding stream. We shall refer to the parameter $\beta_{ik}^{(j)}$ as a *shrinkage factor*, which represents the shrinkage (if < 1) or expansion (if > 1) effect in stream processing. Thus flow conservation may not hold in the processing stage.

It is possible for a stream to travel along different paths to reach the sink. Resource consumption may also vary along the different paths. However, the resulting outcome does not depend on the processing path. This leads to the following assumption on β :

Property 1 For any two distinct paths $p = (n_0, n_1, \dots, n_l)$ and $p' = (n'_0, n'_1, \dots, n'_l')$ that have the same starting and ending points, i.e. $n_0 = n'_0$ and $n_l = n'_l'$, we must have, for any commodity j , $\prod_{i=0}^{l-1} \beta_{n_i, n_{i+1}}^{(j)} = \prod_{i=0}^{l'-1} \beta_{n'_i, n'_{i+1}}^{(j)}$.

For a given node $n \in \mathcal{N}$, define $g_n(j)$ to be the product of the $\beta_{ik}^{(j)}$'s along any path from source s_j to node n . That is, no matter which path it takes, the successful delivery of one unit of commodity j from source s_j to node n results in $g_n(j)$ amount of output at node n . Clearly, $g_{s_j}(j) = 1$. If node n is not reachable from s_j , we also set $g_n(j) = 1$. Hence $g_n(j)$ is always positive and defined for all commodities and all nodes.

Utility Function: Our goal is to design a joint admission control, data routing, and resource allocation mechanism such that the overall *information* delivered by the stream processing network is maximized. We distinguish here between data and information in the following sense. Let a_j be the rate at which data from source s_j is delivered to sink j . A utility function $U_j(a_j)$ quantifies the *value* of this data to the data-consuming applications. We assume that U_j is a concave and increasing function, reflecting the decreasing marginal returns of receiving more data. As discussed below, our goal is to maximize the overall system utility $U = \sum_j U_j(a_j)$, rather than the rate at which data is delivered.

Since the system is constrained in both computing power and communication bandwidth, each server is faced with two decisions: first, it has to allocate its computing power to multiple processing tasks; second, it has to share the bandwidth on each output link among the multiple flows going through it.

Problem Formulation: The problem can be formulated as the following utility optimization problem.

| |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Given: network $G = (\mathcal{N}, \mathcal{E})$, resource budget \mathcal{C}, resource consumption rate c, shrinkage factor β, and data input rate Λ.</p> <p>Maximize: Overall system utility $U = \sum_j U_j(a_j)$.</p> <p>Constraints:</p> <ol style="list-style-type: none"> 1) Per node resource constraint; 2) Per link bandwidth constraint; 3) Flow balance constraints that account for shrinkage factors; 4) $a_j \leq \lambda_j, \quad \forall j$, |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

where a_j denotes the admission rate of commodity j flow at source s_j , $j = 1, \dots, J$.

The flow balance constraints ensure that incoming flows arrive at the same rate as outgoing flows being consumed (so as to be processed) at each node for each commodity. Note that due to the shrinkage and expansion effects, for one unit of commodity j flow on node i heading towards node

k , after processing, it becomes $\beta_{ik}^{(j)}$ units of actual outgoing flow to downstream node k .

3 Problem Transformation

The problem presented above requires the optimal allocation of two different resources (computing power per node and communication bandwidth per link). Moreover, it requires admission control at sources since the optimal injection rate a_j is not known until one solves the optimization problem. In this section, we present ways to unify the different two resources and also transform the joint resource allocation and admission control problem into a tangible routing problem.

Bandwidth Node: We next present a scheme to extend the original graph so that we can address the two different resources (computing power and link bandwidth) in a unified way. We do so by introducing a bandwidth node, denoted as n_{ik} , for each edge $(i, k) \in \mathcal{E}$. We also add directed edges (i, n_{ik}) and (n_{ik}, k) (see Figure 2). We assume that bandwidth node n_{ik} has a total resource $C_{n_{ik}} = B_{ik}$. The role of a bandwidth node is to transfer flows. It requires one unit of its resource (bandwidth) to transfer one unit of flow, which becomes one unit of flow for the downstream node. In other words, $\beta_{n_{ik},k}^{(j)} = 1$, $c_{n_{ik},k}^{(j)} = 1$. In addition, we set $c_{i,n_{ik}}^{(j)} = c_{ik}^{(j)}$, $\beta_{i,n_{ik}}^{(j)} = \beta_{ik}^{(j)}$.

With the addition of the bandwidth nodes (and corresponding links), the original problem of allocating two different resources is transformed a unified resource allocation problem with a single resource constraint on each node. In the new system, each node only has a single resource constraint associated with it. If it is a bandwidth node, then it is constrained by bandwidth; if it is a processing node, then it is constrained by the computing resource. The new system is then faced with a unified problem: finding efficient ways of shipping all J commodity flows to their respective destinations subject to the (single) resource constraints at each node.

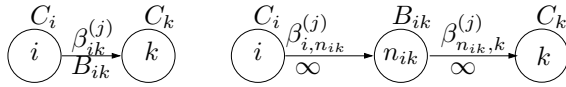


Figure 2. Extended graph.

Dummy Node: An algorithm for the continuous flow problem is *stable* if it is able to deliver in the long run the injected flow at rate a_j at source s_j , $j = 1, \dots, J$. However, the optimal injection rate a_j is not known until one solves the optimization problem. We resolve this by introducing additional dummy nodes and dummy links, and then present an algorithm that determines the optimal rates a_j 's automatically for the continuous problem. To do this, we also need

to transform the capacity constraints in the original problem to the objective function. The addition of dummy nodes is similar to that in [5], which was originally proposed in [11].

For each source node s_j , we introduce a dummy node \bar{s}_j . We also add a dummy input link (\bar{s}_j, s_j) and a dummy difference link (\bar{s}_j, j) as shown in Figure 3. The dummy node \bar{s}_j has no resource constraint, i.e. $C_{\bar{s}_j} = +\infty$.

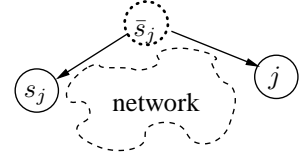


Figure 3. Dummy node.

We make the dummy node \bar{s}_j the new source for commodity j , where traffic of commodity j arrives at node \bar{s}_j at a *fixed* rate λ_j . Node \bar{s}_j sends traffic across link (\bar{s}_j, s_j) at rate a_j , and the remainder of the incoming traffic $\lambda_j - a_j$ across the dummy difference link (\bar{s}_j, j) to sink j . Define the cost of carrying flow x over link (\bar{s}_j, j) to be the utility loss over the link, i.e.

$$Y_{(\bar{s}_j, j)}(x) = U_j(\lambda_j) - U_j(\lambda_j - x). \quad (1)$$

The problem of maximizing the utility $U = \sum_j U_j(a_j)$ is equivalent to minimizing the utility loss over all dummy difference links i.e.

$$\min Y = \sum_j Y_{(\bar{s}_j, j)}(\lambda_j - a_j).$$

Since the utility function U_j is concave and increasing, the cost function Y is convex and increasing.

For convenience, we define $Y_{(i,k)}(x) = 0$ for all other links (other than the dummy difference links). Denote by $r(j)$ the (external) input traffic rate vector for commodity j , where

$$r_i(j) = \begin{cases} \lambda_j & \text{if } i = \bar{s}_j \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

We denote by $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ the resulting new graph, where \mathcal{V} denotes the extended node set (including the bandwidth nodes and dummy nodes) and \mathcal{L} the extended edge set (including the added dummy input links and dummy difference links). Last, for node i , let $L_I(i)$ denote the set of links that terminates at it, $L_U(i)$ the set of links that emanates from it, and $L(i) = L_I(i) \cup L_U(i)$ the set of links adjacent to node i .

Clearly, after the above transformation, an original graph G with N nodes, M edges and J commodities produces a new graph \mathcal{G} with $N + M + J$ nodes, $2M + 2J$ edges and J commodities. We work on the new graph \mathcal{G} from here on.

We next introduce convex and increasing penalty functions to account for the per node resource constraints. For a usage z of resource at node i , a penalty $D_i(z)$ will be incurred. We assume $D_i(z)$ is convex and increasing in z and

$\lim_{z \rightarrow C_i} D_i(z) \rightarrow \infty$, where C_i is the total resource budget at node i . Such a penalty function can be, for example, $D_i(z) = \frac{1}{C_i - z}$. Note that $D_i = 0$ for all dummy nodes since they have infinite capacity.

Let $D = \sum_{i \in \mathcal{N}'} D_i(z_i)$ where z_i denotes the resource usage on node i . The overall system utility then becomes $Y + \epsilon D$, where ϵ is a tunable parameter. With the dummy nodes and resource penalty function, the problem then becomes a routing problem with the objective to minimize the total cost

$$A = Y + \epsilon D.$$

The use of penalty functions results in an allocation that is *not* strictly identical to the optimal solution to the original problem before the penalty function was introduced. However, by selecting ϵ appropriately, this standard approach typically results in a solution that is nearly the optimal solution to the initial problem formulation. A penalty function may also prevent a node resource (or a link capacity) from being completely allocated. In practice, such remaining capacity could be used to better accommodate changing demands, or for faster recovery in the case of node or link failures.

With the above transformation, we now have the following utility optimization problem on the new graph G' .

Given: network $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ resource budget \mathcal{C} , resource consumption rate c , shrinkage factor β , and data input rate Λ .

Minimize: Cost Function $A = Y + \epsilon D$.

Constraints:

- 1) Per node resource constraint;
- 2) Flow balance constraints (need to factor in shrinkage factors);
- 3) $a_j \leq \lambda_j, \quad \forall j$.

4 Distributed Problem Formulation for Joint Routing and Resource Allocation

The continuous version of the above (static) optimization problem can be interpreted as a flow problem in which source \bar{s}_j pumps commodity j flow into the system at rate λ_j . In order to solve this problem using a distributed algorithm, we reformulate the problem using local routing fractions as control variables. The resulting problem then becomes a joint routing and resource allocation problem.

Let $t_i(j)$ denote the total expected traffic rate at node i for commodity j . Let $\phi_{ik}(j)$ denote the fraction of $t_i(j)$ that will be processed over link (i, k) . We call $\phi = \{\phi_{ik}(j) : i, k \in \mathcal{V}, j = 1, \dots, J\}$ the routing decision, if $\phi \geq 0$, $\sum_k \phi_{ik}(j) = 1$ for each non-sink node i and $\phi_{ik}(j) = 0$ if $(i, k) \notin \mathcal{E}$ or i is a sink node for one commodity.

Note that it takes resource $c_{ik}^{(j)}$ from node i to process a unit of commodity j flow, and once across edge (i, k) , it

produces $\beta_{ik}^{(j)}$ unit of commodity j flow due to the shrinkage factor. We have the following flow balance equations:

$$t_i(j) = r_i(j) + \sum_l t_l(j) \phi_{li}(j) \beta_{li}^{(j)}, \quad (3)$$

Equation (3) implicitly expresses the balance of flow at each node accounting for the shrinkage factors: the total flow rate into a node (after the shrinkage/expansion) is equal to the rate out of the node for each commodity j . It can be shown that equation (3) has a unique solution of t given r and ϕ .

Now let f_{ik} be the total expected resource usage rate from node i by all (commodity) flows across edge (i, k) , and f_i the total resource usage rate on node i . We have,

$$f_{ik} = \sum_j t_l(j) \phi_{ik}(j) c_{ik}^{(j)}. \quad (4)$$

$$f_i = \sum_{(i,k) \in L_U(i)} f_{ik}. \quad (5)$$

Clearly, a feasible set of flows f must satisfy the capacity constraints.

$$f_i \leq C_i, \quad i \in \mathcal{V} \quad (6)$$

To account for the shrinkage factor, the flow conservation at each node using flow variable set f is given as follows. For $f_{ik}(j) \geq 0, (i, k) \in E, j \in V$,

$$\sum_{(i,k) \in L_U(i)} f_{ik}(j) - \sum_{(l,i) \in L_I(i)} f_{li}(j) \beta_{li}^{(j)} = r_i(j), \quad i \neq j. \quad (7)$$

We further decompose cost A into node-level local costs. For a given flow set f , the node i cost, $A_i(f)$, for $i \in \mathcal{V}$, is defined as follows:

$$A_i(f) = \epsilon D_i(f_i) + \sum_{(i,k) \in L_U(i)} Y_{(i,k)}(f_{ik}), \quad (8)$$

where $f_i = \sum_{(i,k) \in L_U(i)} f_{ik}$ is the total resource usage rate at node i . Clearly, $A = \sum_{i \in \mathcal{V}} A_i$.

The joint data routing and resource allocation problem is then reformulated using routing variable set ϕ as control variables:

Given: network $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, resource budget \mathcal{C} , resource consumption rate c , shrinkage factor β , and data input rate Λ

Minimize: Cost $A = \sum_i A_i$.

Constraints: Flow set f is implemented by routing variable set ϕ .

5 A Distributed Algorithm for Routing Optimization

In the previous section, for given fixed set of routes, nodes can achieve the optimal resource allocation through

independent node-level resource optimization, and calculate the marginal global cost through local sensitivity analysis and communication between neighbor nodes. Now, we focus on routing optimization. We generalize Gallager's result [10] and propose a distributed routing algorithm that converges to the optimal routing solution.

For given routing decision ϕ and the resulting resource usage rate f , let $A_i^f(f)$ (or $A_i^\phi(\phi)$) denote the cost incurred at node i . Denote by $A^f(f)$ (or $A^\phi(\phi)$) the corresponding total cost. Similar to [10], we compute the partial derivatives of A^ϕ with respect to the inputs r and the routing variables ϕ as follows.

$$\frac{\partial A^\phi(\phi)}{\partial r_i(j)} = \sum_k \phi_{ik}(j) \left[\frac{\partial A_i^f(f)}{\partial f_{ik}} c_{ik}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_k(j)} \beta_{ik}^{(j)} \right], \quad (9)$$

$$\frac{\partial A^\phi(\phi)}{\partial \phi_{ik}(j)} = t_i(j) \left[\frac{\partial A_i^f(f)}{\partial f_{ik}} c_{ik}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_k(j)} \beta_{ik}^{(j)} \right], \quad (10)$$

where based on (8), (5) and (1),

$$\frac{\partial A_i^f(f)}{\partial f_{ik}} = \begin{cases} U'_k(\lambda_k - f_{ik}) & \text{if } i = \bar{s}_j \text{ and } k = j \\ \epsilon D'_i(f_i) & \text{else} \end{cases} \quad (11)$$

One can further show the following necessary and sufficient conditions to minimize A^ϕ over all feasible sets of routes.

Theorem 2 *Let F be a convex and compact set of flow sets, which is enclosed by $|E|$ planes (each of which corresponds to $f_{ij} = 0$, $(i, j) \in E$), and a boundary envelope F_∞ . Assume that A^f is convex and continuously differentiable for $f \in F \setminus F_\infty$. Let Ψ be the set of ϕ for which the resulting set of flow rates f lie in set $F \setminus F_\infty$. Then the necessary (but not sufficient) condition for ϕ to minimize A^ϕ over Ψ is that, for all $i \neq j$, $(i, k) \in E$:*

$$\frac{\partial A^\phi(\phi)}{\partial \phi_{ik}(j)} \begin{cases} = \lambda_{ij} & \phi_{ik}(j) > 0 \\ \geq \lambda_{ij} & \phi_{ik}(j) = 0. \end{cases} \quad (12)$$

The sufficient condition is that, for all $i \neq j$, $(i, k) \in E$,

$$\frac{\partial A_i^f(f)}{\partial f_{ik}} c_{ik}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_k(j)} \beta_{ik}^{(j)} \geq \frac{\partial A^\phi(\phi)}{\partial r_i(j)}. \quad (13)$$

Based on the above sufficient condition, we now develop a gradient-based algorithm by generalizing the algorithm presented in [10]. Each node i must incrementally decrease those routing variables $\phi_{ik}(j)$ for which the marginal cost $\partial A_i^f(f)/\partial f_{ik} c_{ik}^{(j)} + \partial A^\phi(\phi)/\partial r_k(j) \beta_{ik}^{(j)}$ is large, and increase those for which it is small. The algorithm divides into three components: a protocol between nodes to calculate the marginal costs, an algorithm for calculating the routing updates and modifying the routing variables, and a protocol for forecasting the flow rates of next iteration and allocating resources to support them. We discuss the protocol to calculate the marginal costs first.

Let us see how node i can calculate $\partial A^\phi(\phi)/\partial r_i(j)$. Define node m to be downstream from node i (with respect to destination j) if there is a routing path from i to j passing through m (i.e., a path with positive routing variables on each link). Similarly, we define i as upstream from m if m is downstream from i . A routing variable set ϕ is loop free if for each destination j , there is no $i, m (i \neq m)$ such that i is both upstream and downstream for m . The protocol used for an update, now, is as follows: for each destination node j , each node i waits until it has received the value $\partial A^\phi(\phi)/\partial r_m(j)$ from each of its downstream neighbors $m \neq j$. Node i then calculates $\partial A^\phi(\phi)/\partial r_k(j)$ from (9) (using the convention that $\partial A^\phi(\phi)/\partial r_j(j) = 0$) and broadcasts this to all of its neighbors. It is easy to see that this procedure is deadlock-free if and only if ϕ is loop free.

We shall later define a small but important detail that has been omitted so far in the update protocol between nodes: a small amount of additional information is necessary for the algorithm to maintain loop freedom. It is necessary, for each destination j and each node i , to specify a set $B_i(j)$ of blocked nodes k for which $\phi_{ik}(j) = 0$ and the algorithm is not permitted to increase $\phi_{ik}(j)$ from 0. We first define and discuss the algorithm and then define the sets $B_i(j)$.

The algorithm Γ , on each iteration, maps the current routing variable set ϕ into a new set $\phi^1 = \Gamma(\phi)$. The mapping is defined as follows. For $k \in B_i(j)$,

$$\phi_{ik}^1(j) = 0, \quad \Delta_{ik}(j) = 0. \quad (14)$$

For $k \notin B_i(j)$, define

$$a_{ik}(j) = \frac{\partial A_i^f(f)}{\partial f_{ik}} c_{ik}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_k(j)} \beta_{ik}^{(j)} - \min_{m \notin B_i(j)} \left[\frac{\partial A_i^f(f)}{\partial f_{im}} c_{im}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_m(j)} \beta_{im}^{(j)} \right] \quad (15)$$

$$\Delta_{ik}(j) = \min[\phi_{ik}(j), \eta a_{ik}(j)/t_i(j)] \quad (16)$$

where η is a scale parameter of Γ to be discussed later. Let $\underline{k}(i, j)$ be a value of m that achieves the minimization in (16). Then

$$\phi_{ik}^1(j) = \begin{cases} \phi_{ik}(j) - \Delta_{ik}(j) & k \neq \underline{k}(i, j) \\ \phi_{ik}(j) + \sum_{k \neq \underline{k}(i, j)} \Delta_{ik}(j) & k = \underline{k}(i, j). \end{cases} \quad (17)$$

The algorithm reduces the fraction of traffic sent on non-optimal links and increases the fraction on the best link. The amount of reduction, given by $\Delta_{ik}(j)$, is proportional to $\alpha_{ik}(j)$, with the restriction that $\phi_{ik}^1(j)$ cannot be negative. In turn $\alpha_{ik}(j)$ is the difference between the marginal cost to node j using link (i, k) and using the best link. Note that, as condition (13) is approached, the changes become smaller, as desired. The amount of reduction is also inversely proportional to $t_i(j)$. The reason for this is that the change in link traffic is related to $\Delta_{ik}(j)t_i(j)$. Thus when $t_i(j)$ is

small, $\Delta_{ik}(j)$ can be changed by a large amount without greatly affecting the marginal cost. Finally the changes depend on the scale factor η . For η very small, convergence of the algorithm is guaranteed, but rather slowly. As η increases, the speed of convergences increases but the danger of no convergence increases. In the next section, we identify particular values of η through simulation.

We now complete the definition of algorithm Γ by defining the block sets $B_i(j)$. See [10] for further reasoning on how this definition guarantees the loop free properties.

Definition: The set $B_k(j)$ is the set of nodes k for which both $\phi_{ik}(j) = 0$ and k is blocked relative to destination j . A node k is blocked relative to j if k has a routing path to j containing some link (l, m) for which $\phi_{lm}(j) > 0$, and $\partial A^\phi(\phi)/\partial r_l(j) \leq \partial A^\phi(\phi)/\partial r_m(j)$, and

$$\phi_{lm}(j) \geq \frac{\eta}{t_i(j)} \left[\frac{\partial A_l^f(f)}{\partial f_{lm}} c_{lm}^{(j)} + \frac{\partial A^\phi(\phi)}{\partial r_m(j)} \beta_{lm}^{(j)} - \frac{\partial A^\phi(\phi)}{\partial r_l(j)} \right]. \quad (18)$$

The protocol required for a node i to determine the set $B_i(j)$ is as follows. Each node l , when it calculates $\partial A^\phi(\phi)/\partial r_l(j)$, determines, for each downstream m , if $\phi_{lm}(j) > 0$, and $\partial A^\phi(\phi)/\partial r_l(j) \leq \partial A^\phi(\phi)/\partial r_m(j)$, and satisfy (18). If any downstream neighbor satisfies these conditions, node l adds a special tag to its broadcast of $\partial A^\phi(\phi)/\partial r_l(j)$. The node l also adds the special tag if the received value $\partial A^\phi(\phi)/\partial r_m(j)$ from any downstream m contained a tag. In this way all nodes upstream of l also send the tag. The set $B_i(j)$ is then the set of nodes k for which the received $\partial A^\phi(\phi)/\partial r_k(j)$ was tagged.

Finally, we describe the protocol for forecasting the flow rates for the next iteration and allocating resources to support the updated traffic. Assume that each node i can estimate the demand rate set $r_i(j)$ entering from i . First, for each destination node j , each node i signals the downstream nodes under ϕ^1 (the set of routes for the next iteration) so that each node k gets a list of upstream nodes under ϕ^1 . Second, for each destination node j , each node i waits until it has received the forecasted value $f_{li}^1(j)$ from each of its upstream nodes, l , under ϕ^1 . For each downstream node k under ϕ^1 , each node i then calculates $f_{ik}^1(j)$ from (3)(4) and sends it to k . Node i also calculates forecasted f_i^1 , from (5). Based on the forecasted link data rates of incoming and outgoing links f^1 , each node finds locally the resource allocation by minimizing its node-level cost function.

We have proposed a distributed algorithm for routing optimization. Note that in each iteration, the resource allocation is also optimized through local independent resource optimization at all nodes. Combining the collective routing optimization, and independent local resource optimization at all nodes, we have achieved the optimal cost over all feasible resource allocation and routing combinations.

6 Numerical Examples

In this section, we illustrate through a particular example the convergence speed of the proposed algorithm. For convenience, we refer to it as the gradient-based algorithm. We compare the performance with that of the back-pressure algorithm presented in our earlier work [6].

Here we briefly review the back-pressure algorithm proposed in [6]. Each node maintains local input and output buffers for each commodity. Each node also maintains a potential function. The algorithm is iterative in nature and, at each iteration, a node only needs to know the buffer levels at its neighboring nodes. It then uses this information to determine the appropriate resource allocation that reduces the potential at that node by the greatest amount. This local control mechanism can be shown to lead to the global optimal solution.

We apply both the gradient-based algorithm and the back-pressure algorithm on a synthetic (random) network containing 40 nodes, and 3 source and sink pairs corresponding to a 3-commodity problem. The system utility is taken to be the total throughput of the 3 commodities. Both the link capacities and node computing capacities are generated from independent uniform random samples in the range $[1, 100]$. The g_n^j parameters are real numbers uniformly distributed in $[1, 10]$, from which we then obtain the shrinkage parameter by setting $\beta_{ik}^j = \frac{g_k^j}{g_i^j}$ based on Property 1. The resource consumption parameters r are real numbers uniformly distributed in $[1, 5]$.

The red curve in Figure 4 shows the system throughput achieved by the gradient-based algorithm as a function of the number of iterations in log-scale. The horizontal line represents the optimal throughput obtained using an optimization solver. Here we set the penalty cost coefficient $\epsilon = 0.2$, scale factor $\eta = 0.04$, we see that about 1000 iterations are required to achieve a utility that is within 95% of optimal. As we mentioned in the previous section, the choice of the scale factor η has great impact the convergence speed. With a small η , the algorithm will eventually converge to the optimum but at a slow rate. In practice, it is possible to choose a η much larger to expedite the convergence, e.g. in hundreds of iterations.

The green curve in Figures 4 shows the performance under the back-pressure algorithm. Observe that for both algorithms, the total throughput improves monotonically until it eventually reaches the optimum. The number of iterations required by the gradient-based algorithm is on the scale of hundreds or thousands (depending on the choice of η), while the back-pressure algorithm requires almost 100,000 iterations to reach within 95% of optimal. The gradient-based algorithm is therefore more efficient in number of iterations as each time it tries the steepest decent for the overall utility function.

It is unfair, however, to compare the two algorithms solely based on the number of iterations given that the two algorithms do completely different things during each iteration. An iteration in the gradient-based algorithm is generally more expensive since each node needs to wait until it has received the value $\partial A^\phi(\phi)/\partial r_m(j)$ from each of its downstream neighbors $m(\neq j)$ for each destination node j in order to update its own value of the partial derivative. This can be time consuming. It takes $O(L)$ number of message exchanges to update all nodes, where L represents the length of the longest path in the network. An iteration in the back-pressure algorithm is much faster. Each node simply exchanges the buffer levels with its neighboring nodes and then makes the resource allocation decision locally. All nodes do this in parallel and independently, and it takes just $O(1)$ number of message exchanges. Therefore, the gradient-based algorithm may be better when the depth of the graph is not large, or else the back-pressure algorithm may be favored. Further study is needed to more carefully determine the conditions under which each algorithm is fastest to converge.

7 Conclusion

In summary, we have studied the problem of how to distribute the processing of a variety of data streams over the multiple cooperative servers in a communication network. The network is resource constrained in both computing power at each server and in bandwidth capacity over the various communication links. We present a graph representation of the problem and show how to map the original problem into an equivalent multicommodity flow problem. We have developed distributed algorithms and presented both theoretical analysis and numerical experiments. We show that the proposed distributed algorithms are able to achieve the optimal solution in the long run.

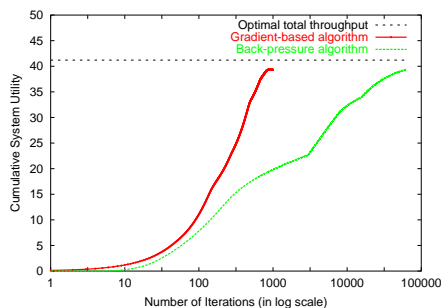


Figure 4. Comparison of Gradient-based algorithm with Back-pressure algorithm.

References

- [1] D. J. Abadi et al. The design of the borealis stream processing engine. In *Proc. of CIDR*, pages 277–289, 2005.
- [2] B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. of FOCS*, pages 459–468, 1993.
- [3] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Load management and high availability in the medusa distributed stream processing system. In *Proc. of SIGMOD*, pages 929–930, 2004.
- [4] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 1977.
- [5] D. Bertsekas and R. Gallager. *Data networks*. Prentice-Hall, 1987.
- [6] J. Broberg, Z. Liu, C. H. Xia, and L. Zhang. A multi-commodity flow model for distributed streaming processing. In *Proc. of SIGMETRICS*, 2006.
- [7] S. Chandrasekaran and M. J. Franklin. Remembrance of streams past: Overload-sensitive management of archived streams. *30th VLDB*, 2004.
- [8] L. Chen, K. Reddy, and G. Agrawal. Gates: A gridbased middleware for processing distributed data streams. In *Proc. of HPDC*, 2004.
- [9] M. Cherniack et al. Scalable distributed stream processing. In *Proc. of CIDR*, 2003.
- [10] R. Gallager. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications*, pages 73–85, 1977.
- [11] R. Gallager and S. J. Colestaani. Flow control and routing algorithms for data networks. *Proc. Int. Conf. on Computer Commun.*, pages 779–784, 1980.
- [12] N. Jain, L. Amini, H. Andrade, R. King, Y. Park, P. Selo, and C. Venkatramani. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *Proc. of SIGMOD*, pages 431–442, 2006.
- [13] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices proportional fairness and stability. *Journal of the Operational Research Society*, 1998.
- [14] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proceedings of PODS*, pages 250–258, 2005.
- [15] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proc. of 29th VLDB Conf.*, pages 309–320, 2003.