

Utility Optimization for Event-Driven Distributed Infrastructures

Cristian Lumezanu
University of Maryland, College Park
lume@cs.umd.edu

Sumeer Bhola, Mark Astley
IBM T.J. Watson Research Center
{sbhola, mastley}@us.ibm.com

Abstract

Event-driven distributed infrastructures are becoming increasingly important for information dissemination and application integration. We examine the problem of optimal resource allocation for such an infrastructure composed of an overlay of nodes. Resources, like CPU and network bandwidth, are consumed by both message flows and message consumers; therefore, we consider both rate control for flows and admission control for consumers. This makes the optimization problem difficult because the objective function is nonconcave and the constraint set is nonconvex. We present LRGP (Lagrangian Rates, Greedy Populations), a scalable and efficient distributed algorithm to maximize the total system utility. The key insight of our solution involves partitioning the optimization problem into two types of subproblems: a greedy allocation for consumer admission control and a Lagrangian allocation to compute the flow rates, and linking the subproblems in a manner that allows tradeoffs between consumer admission and flow rates while satisfying the nonconvex constraints. LRGP allows an autonomic approach to system management where nodes collaboratively optimize aggregate system performance. We evaluate the quality of results and convergence characteristics under various workloads.

1. Introduction

Event-driven distributed infrastructures are becoming increasingly important and span middleware technologies such as content-based publish/subscribe [4, 1], stream processing overlays [30, 19, 28], and Enterprise Service Bus (ESB) [5]. The common theme across these technologies is support for asynchronous, loosely-coupled event/message delivery, including transformations which may alter messages as they flow from producers to consumers. Examples of message transformation include filtering based on content, format changes for integrating legacy applications, augmenting messages with content retrieved from databases, and aggregating multiple messages to produce a more concise stream. Furthermore, these infrastructures

need to support applications that have a heterogeneous set of requirements. For instance, an ESB seeks to integrate all applications within an enterprise including federations of applications across partner enterprises.

The workloads placed on an event-driven distributed infrastructure (referred to as the *system* from now on) can be bursty, typically because communication is triggered by real world events. In the presence of significant workload fluctuation, over-provisioning resources to meet peak load requirements is not desirable since it has significant cost in terms of hardware, space, power and human resources. Instead, it is preferable to appropriately manage the dynamic allocation of existing resources.

In this paper, we consider the problem of optimal resource allocation in an event-driven distributed infrastructure consisting of an overlay of computing nodes. The problem allows both rate control and admission control, and tradeoffs between adjusting the rate of messages and admitting consumers. Our distributed solution enables the nodes of the system to collaboratively optimize aggregate system performance, hence it is a self-optimization scheme that is applicable to an autonomic event-driven infrastructure. Next, we give an overview of the problem.

1.1. Problem Overview

We illustrate the problem with a few simple examples:

- *Trade Data*: An application is producing messages corresponding to each trade in the stock market for a certain market segment. There are two kinds of consumers interested in this data: (1) consumers at one or more brokerage firms, called *gold* consumers, which pay for the data, and (2) *public* consumers connected over the Internet. The gold consumers have a higher priority because they bring more benefit to the system. A consumer attaches to a node in the system, and receives the messages it is interested in. Before being provided to public consumers, messages are altered within the system and certain fields available only to gold consumers are removed. In addition, gold consumers expect reliable and fast delivery, which places extra overhead on the system to process acknowledge-

ments and to control the *inelastic* rate (*i.e.* cannot be decreased to tolerate delays [27]). In case of lack of resources, the system can reduce load by using admission control to deny service to public consumers.

- **Latest Price Data:** An application is producing messages representing the latest prices of IBM stock. Public consumers connected to the system receive price messages which satisfy a consumer-specified filter (*e.g.*, price > 80). That is, for each price message, the system evaluates the filter to determine whether the message should be delivered to the consumer. The message flow is very elastic, since rate can be decreased (and latency increased) by reducing the frequency of updates. In case of lack of resources, the system can reduce load by either reducing the producer rate or denying service to consumers or both.

In the above scenarios, system resources are consumed both on a per message basis, independent of the number of consumers, and on a per message, per consumer basis. The cost of the latter can vary depending on the complexity of the per consumer processing, like content filtering or ensuring reliable delivery.

These and other scenarios can be generalized as follows. Producer messages injected into the system are classified into flows. Messages may be transformed as they are disseminated from producers to consumers. That is, while flows consume bandwidth resources at each link they traverse, they also consume CPU resources at nodes that transform and route messages. Consumers of a flow are organized into consumer classes, with one or more consumer classes per flow. Admission control is used to adjust the number of consumers for each class.

We use the concept of utility functions to describe the benefit of the system. Utilities are expressed in terms of message rates¹ and are associated with consumer classes such that all consumers in a class share the same utility function. More precisely, if the system has n_j consumers of class j currently receiving messages from flow i , then the aggregate utility of these consumers is $n_j \times U_j(r_i)$, where r_i is the rate at which messages corresponding to flow i are being injected into the system, and U_j is the utility function associated with class j . We assume U_j is strictly concave. A resource allocation is optimal when it maximizes the overall system utility (*i.e.*, the sum of the aggregate utilities for each class). The optimizer determines the rate of messages for each flow, and the set of consumers admitted to each consumer class.

1.2. Solution Overview

In general, the optimization problem is difficult to solve because the utility of the system depends both on the rate

¹Equivalently, we can define utility as a function of latency since changes in message rate directly correspond to changes in latency.

allocation (*i.e.*, rates of all the flows) and on the consumer allocation (*i.e.*, the number of consumers admitted). This makes the objective function (*i.e.*, the sum of the aggregate consumer utilities) not concave, despite U_j being concave, and the resource allocation constraints at the nodes nonconvex, as we will explain in Section 2. In contrast, related work on network flow optimization [23, 16] considers only rate allocation, and typically maximizes a concave objective function on a convex set. For scalability, we would also like a distributed solution, where a flow rate is being decided at its source, and consumer admission is being decided at each node.

The main contribution of this paper is LRGP (Lagrangian Rates, Greedy Populations), a scalable and efficient distributed algorithm for maximizing the total utility in an event-driven distributed infrastructure. The key idea of our solution involves partitioning the optimization problem into two types of subproblems: (1) a greedy approach that admits consumers based on a total ordering of their benefit-cost ratio at each node, and (2) a Lagrangian approach to compute each flow rate. Subproblem (1) generates prices used in subproblem (2) and allows tradeoffs between flow rates and consumer admission. To satisfy the nonconvex constraints, subproblem (1) turns them into convex constraints and attempts to satisfy them locally at each node. We construct various workloads to evaluate the quality of the results generated by the algorithm and its convergence characteristics, and present a detailed experimental evaluation. To the best of our knowledge this is the first algorithm to solve the problem.

The rest of the paper is organized as follows. In the next section, we formally define the problem statement. In Section 3, we describe the LRGP algorithm. We evaluate LRGP by way of several experiments in Section 4. In Section 5 we review related work. We conclude in Section 6.

2. Problem Specification

We begin this section with a description of the system model, followed by the specification of the objective function and resource constraints, and conclude with a discussion of the problem specification.

2.1. System Model

Consider a network of nodes interconnected by links. Links are unidirectional, and have a fixed capacity associated with them. Each node also has a capacity. Producers and consumers connect to the nodes of the system. A producer publishes messages on one flow, and all the producers publishing to a particular flow connect to the same node, called the *source node* for the flow.

A consumer that connects to a node is interested in receiving messages from a particular flow. However, it does

not receive service until it is admitted by the node. An unadmitted consumer remains connected to the node until it is admitted, and disconnects when it is no longer interested in receiving service. A consumer that has been admitted can be unadmitted, possibly because sufficient resources are no longer available in the system to satisfy its needs. The system exercises resource control in two ways: (1) the rate allocation to a flow at its source node; and, (2) admission control on consumers.

The link capacity can be considered to be equal to the network bandwidth available on the path between the two nodes connected by a link, and node capacity can be a measure of the available CPU capacity at the machine (or set of machines) that constitutes a node.

In subsequent sections, we describe the algorithm in a manner such that it is running all the time, and responding to changes in workload and system capacity by adjusting rates and consumer allocation. However, in a real system setting, making very frequent admission control decisions may be disruptive to consumers using the system, so the decisions may not be *enacted* until their values are sufficiently different from the previous enacted values, or may be enacted periodically (say once every few minutes).

2.2. Objective Function

Let \mathcal{F} be the set of all flows, \mathcal{C} be the set of all consumer classes, \mathcal{B} be the set of all nodes, and let \mathcal{L} be the set of all links. Each consumer class is associated with at most one flow. The set $C_i \subset \mathcal{C}$ denotes the set of consumer classes associated with flow i . For convenience, we define a function $\text{flowMap} : \mathcal{C} \rightarrow \mathcal{F}$ such that $\text{flowMap}(j) = i$ if $j \in C_i$.

The number of admitted consumers in a class $j \in \mathcal{C}$ is denoted by n_j and all of them share the same utility function $U_j(r_i)$, where r_i is the rate of the flow associated with the class j . For simplicity of notation, and without loss of generality, we assume all consumers in a class connect to the same node. In particular, any class which spans multiple nodes can be partitioned into disjoint classes with identical utility functions. For each flow i , we define a function $\text{attachMap}_i : \mathcal{B} \rightarrow 2^{\mathcal{C}}$ which gives the set of classes for flow i which attach to node b . That is, if $\text{attachMap}_i(b) = D$ then each $j \in D$ attaches to node b . Likewise, we define a function $\text{nodeClasses} : \mathcal{B} \rightarrow 2^{\mathcal{C}}$ such that $\text{nodeClasses}(b)$ gives the set of classes (for any flow) which attach to node b .

Our objective is to maximize the total utility, *i.e.*,

$$\max \sum_{i \in \mathcal{F}} \sum_{j \in C_i} n_j U_j(r_i) \quad (1)$$

There are two simple constraints:

$$0 \leq n_j \leq n_j^{\max}, \forall j \quad (2)$$

$$r_i^{\min} \leq r_i \leq r_i^{\max}, \forall i \quad (3)$$

The first constraint ensures that the number of consumers admitted does not exceed the number of consumers that want to receive service for each class, n_j^{\max} . The second constraint ensures some bounds on the rate allocation.

We assume the flows are elastic [27] within the rate bounds, and that $U_j(r_i)$ is an increasing, strictly concave and continuously differentiable function of the rate r_i , $r_i^{\min} \leq r_i \leq r_i^{\max}$.

2.3. Resource Constraints

In addition to the simple constraints, there are constraints due to finite resources, which could be network or CPU bandwidth. We model these constraints using three cost values:

1. Link cost, $L_{l,i}$ is the amount of resource used in link l , per unit rate of flow i . If flow i does not traverse link l , this value is 0. For convenience, we define a function $\text{linkMap} : \mathcal{L} \rightarrow 2^{\mathcal{F}}$ such that $\text{linkMap}(l)$ gives the set of flows which traverse link l . We also define \mathcal{L}_i as the set of links traversed by flow i .
2. Flow-node cost, $F_{b,i}$ is the amount of resource used in node b , per unit rate of flow i , which does not depend on the number of consumers. If flow i does not reach node b , this value is 0. For convenience, we define a function $\text{nodeMap} : \mathcal{B} \rightarrow 2^{\mathcal{F}}$ such that $\text{nodeMap}(b)$ gives the set of flows which reach node b . We also define \mathcal{B}_i as the set of nodes reached by flow i .
3. Consumer-node cost, $G_{b,j}$ is the amount of resource used in node b , per consumer of class j admitted at this node, per unit rate of flow $\text{flowMap}(j)$.

The constraint equations for links and nodes respectively are:

$$\sum_{i \in \text{linkMap}(l)} L_{l,i} r_i \leq c_l \quad (4)$$

$$\sum_{i \in \text{nodeMap}(b)} (F_{b,i} r_i + \sum_{j \in \text{attachMap}_i(b)} G_{b,j} n_j r_i) \leq c_b \quad (5)$$

$$\forall l \in \mathcal{L}, \forall b \in \mathcal{B}$$

where c_l, c_b are the resource capacities of link l and node b respectively. These equations are validated using experiments on the Gryphon system [1]. For a more complete description of the resource model, see [3].

2.4. Discussion

We note the following key points:

1. The resource constraint equations use the flow rate at the source. Links and nodes within the system may observe a flow rate different than that at the source. However, this observed flow rate will be a function of the source rate. Therefore, we can view the coefficients of r_i as compensating for this change and also translating the rate into the amount of resource used.

- The constraint equations implicitly assume that the value of n_j does not impact the coefficients, $L_{l,i}$ and $F_{b,i}$. In a real system, if n_j is chosen to be 0 (even though $n_j^{max} > 0$), a flow may not be routed to some parts of the system. Instead of explicitly incorporating this dependency, we propose the following two-stage approximation (1) solve the above problem assuming that flow i is routed to every node which hosts a class $j \in \mathcal{B}_i$ with $n_j^{max} > 0$, (2) prune the paths where $n_j = 0$ in the previous optimization, by setting certain coefficients $L_{l,i}$, $F_{b,i}$ to 0 and solve the problem. We focus on the first stage in this paper.

3. Optimization Algorithm

In this section we present LRGP (Lagrangian Rates, Greedy Populations), a distributed algorithm that optimizes the total system utility. We start by revealing the intuition which lead us to the solution; then we describe the algorithm in detail and discuss its characteristics.

The intuition which stands at the base of our approach is the following. At any moment, we can make the utility of a class larger either by increasing the rate of the flow associated with the class, or by increasing the number of consumers of the class. Increasing the number of consumers has a localized effect on the system: only the resource of the node where the consumers are attached is affected. As such, the node can make a local decision on how much to increase the consumer number. On the other hand, increasing the rate of the flow can potentially affect the resource constraints of all the nodes and links on the flow's path. In order to guarantee that none of these constraints are violated, the source node would have to be coordinated with all the other source nodes, which is impractical in real systems [23]. To solve the problem in a distributed setting we use the concept of price [16, 23]. A price is associated with each link and node resource and indicates how congested the resource is. Each node and link computes a price value² and sends it to the source nodes of the flows that traverse them, which in turn, compute new rates for these flows. A node resource is affected by both the rates of the flows that visit the node and the number of consumers attached to the node. This non-convex constraint cannot be satisfied purely using a price mechanism, so the consumer allocation at a node explicitly tries to satisfy the constraint and uses the consumers that were not admitted to set its price. This makes the tradeoff between increasing the rate and increasing the number of consumers.

LRGP divides the optimization problem into two sub-problems: rate allocation and consumer allocation. Rate allocation finds the optimal rate at a certain time, given a

²Link price is actually computed by one of the two nodes which are the endpoints of the link.

constant number of consumers; consumer allocation finds an optimal number of consumers for each class, given constant flow rates and computes new values for link and node prices. LRGP iterates continuously between these two allocations to increase the value of the objective function.

The key ideas in our approach are:

- Rate/Population Separation:** The main optimization problem is divided into two sub-problems which alternately compute rate and population as described above.
- Greedy Population Allocation:** Flow rates are fixed and populations are computed using a greedy approach at each node such that the maximum utility is obtained without violating the node constraint.
- Lagrangian Rate Allocation:** Populations are fixed and rates are computed using a Lagrangian approach at the source of each flow, constraining rates with link and node "prices" rather than with constraint functions.
- Benefit/Cost Price Determination:** A node price is reset after consumer allocation to reflect the value of consumers that were not admitted. This price captures tradeoffs between increasing the number of consumers versus increasing the rates.

A single iteration of LRGP consists of a rate allocation step, a consumer allocation step, and link and node price computation to connect the two steps. LRGP iterates indefinitely but the allocations may only be enacted periodically (e.g., every few minutes, or when significant changes occur).

The rate and consumer allocation and the price computation are described in more detail below.

3.1. Rate Allocation

The rate allocation algorithm runs individually at the source node of each flow. It computes a new rate for the flow, based on the feedback sent by the nodes and links visited by the flow. It uses the Lagrangian [2] of the original optimization problem (Equation 1), assuming the n_j values are constants:

$$L(r_i, p_b, p_l) = \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}_i} n_j U_j(r_i) - \sum_{l \in \mathcal{L}} p_l \left(\sum_{i \in \text{linkMap}(l)} L_{l,i} r_i - c_l \right) - \sum_{b \in \mathcal{B}} p_b \left(\sum_{i \in \text{nodeMap}(b)} \left(F_{b,i} r_i + \sum_{j \in \text{attachMap}_i(b)} G_{b,j} n_j r_i \right) - c_b \right) \quad (6)$$

where p_l and p_b are the Lagrange multipliers and can be interpreted as the price per unit of resource at link l and node b , respectively. We will refer to p_l and p_b simply as prices from now on.

Instead of solving the original optimization problem, the Lagrangian can be used to solve the following alternative problem for each flow i , given specific values for n_j , p_l and p_b [2, 23].

$$D(p_l, p_b) = \max_{r_i} \left(\sum_{j \in \mathcal{C}_i} n_j U_j(r_i) - r_i (P L_i + P B_i) \right) \quad (7)$$

where

$$PL_i = \sum_{l \in \mathcal{L}_i} L_{l,i} p_l \quad (8)$$

$$PB_i = \sum_{b \in \mathcal{B}_i} (F_{b,i} + \sum_{j \in \text{attachMap}_i(b)} G_{b,j} n_j) p_b \quad (9)$$

Since the objective function in Equation 7 depends only on r_i and $U_j(r_i)$ is strictly concave for all $j \in C_i$, the objective function is strictly concave and differentiable. The maximum is found by setting its derivative with respect to r_i to 0.

The rate allocation step is performed by executing the following algorithm at the source node of each flow:

Algorithm 1 Flow Source Algorithm

Output: Rate r_i , for flow i , at iteration $t = 1, 2, \dots$

- 1: Receive the number of allocated consumers $n_j, \forall j \in C_i$ and price values $p_b, p_l, \forall b \in \mathcal{B}_i, \forall l \in \mathcal{L}_i$.
 - 2: Compute a new transmission rate r_i by setting the derivative wrt r_i of the objective function in Equation 7 to 0.
 - 3: Send r_i to all the nodes and all the links in the path of the flow.
-

3.2. Consumer Allocation

The consumer allocation step consists of allocating consumers for each flow. Each consumer-hosting node b runs a consumer allocation algorithm that, given the current rates r_i of the flows through the node, computes values for n_j . The new n_j allocations will be forwarded to the relevant source nodes for the next iteration once new prices have been computed.

We use a greedy approach which sorts the consumer classes at the node in decreasing order of *benefit-cost* ratio. The *benefit-cost* ratio of a consumer class j is the increase in utility divided by the increase in node resource consumed, when n_j is increased by 1. The equation that computes this ratio for class j is:

$$BC_j = \frac{U_j(r_{\text{flowMap}(j)})}{G_{b,j} r_{\text{flowMap}(j)}} \quad (10)$$

Note that the right hand side of this equation is constant since the r_i values are constant, and does not depend on n_j . The greedy algorithm starts with all n_j equal to 0, and increases the n_j of the consumer class with the highest BC_j , until either $n_j = n_j^{\text{max}}$, or the node constraint is reached. In the former case, we continue the allocation with the class with the next highest BC_j . Note that it is possible that all rates exceed the node constraint before allocation occurs; in this case, all n_j remain at 0.

The node consumer allocation step is included in the description of node price determination in the next section.

3.3. Node Price Computation

We define the benefit-cost ratio for node b at iteration t :

$$BC(b, t) = \max_j (BC_j(t)) \quad (11)$$

$$\forall j \in \text{nodeClasses}(b), n_j < n_j^{\text{max}}$$

where $BC_j(t)$ are the individual benefit-cost ratios used in the consumer allocation algorithm at node b at iteration t .

Intuitively, the benefit-cost ratio for the node represents the maximum increase in utility achievable by increasing the consumer allocation for a class that has not achieved its full allocation if the constraint is relaxed by one unit. We would like to use this to represent the price of the node, since it allows us to coordinate and make tradeoffs between increasing the rate and increasing the number of consumers. However, we do not directly use $BC(b, t)$ as the price for the following reasons:

1. The variation in $BC(b, t)$ between iterations can be very high, which can cause instability. Instead we use a dampening scheme which incrementally approaches this value.
2. There is a boundary case where all the n_j values are 0, but the value of $BC(b, t)$ is too low to constrain the rates. In this case, there is no tradeoff to be made between rates and consumers at this node, and the price is only used to tradeoff between rates and to meet the node constraint, c_b .

Let $used_b(t)$ represent the amount of node resource used at the end of the consumer allocation at time t . The price is adjusted using the following formula:

$$p_b(t+1) = \begin{cases} p_b(t) + \gamma_1 (BC(b, t) - p_b(t)), & \text{if } used_b(t) \leq c_b \\ p_b(t) + \gamma_2 (used_b(t) - c_b), & \text{if } used_b(t) > c_b \end{cases} \quad (12)$$

where γ_1 and γ_2 are stepsizes, with $\gamma_1, \gamma_2 \in [0, \infty)$. In Section 4, we will show that the values of γ_1 and γ_2 can be constrained to a much smaller interval.

The following algorithm is run by each node:

Algorithm 2 Node Algorithm

Output: Consumer allocations $n_j, \forall j \in \text{nodeClasses}(b)$ and price p_b , for node b , at iteration $t = 1, 2, \dots$

- 1: Receive the computed rates of the flows that go through b in the current iteration.
 - 2: Compute n_j for each class $j \in \text{nodeClasses}(b)$, using the greedy approach.
 - 3: Compute a new price p_b based on Equation 12.
 - 4: Send the price p_b and the number of allocated consumers for each class n_j to all the source nodes of the flows that reach node b .
-

3.4. Link Price Computation

We use the price adjustment algorithm described by Low *et al.* [23] to compute the link prices. The algorithm is based on the gradient projection method; the link prices are

adjusted in a direction opposite to the gradient of the objective function of the dual problem (Equation 7). The component of the gradient corresponding to the price p_l , $\frac{\partial D}{\partial p_l}$ is the available resource at link l .

The resulting formula for adjusting the link price is:

$$p_l(t+1) = p_l(t) + \gamma_l \left(\sum_{i \in \text{linkMap}(l)} L_{l,i} r_i(t) - c_l \right) \quad (13)$$

where γ_l is a stepsize, with $\gamma_l \in [0, \infty)$.

If the amount of consumed resources exceeds the capacity of the link, then the price associated with the link will increase. Otherwise, the price is reduced, since higher rates can be accommodated. The following algorithm is run on behalf of each link in the system, by one of the two nodes connected by the link:

Algorithm 3 Link Algorithm

Output: Price p_l , for link l , at iteration $t = 1, 2, \dots$

- 1: Receive the computed rates of the flows that go through l in the current iteration.
 - 2: Compute a new price p_l based on Equation 13.
 - 3: Send the price p_l to all the source nodes of the flows that reach node b .
-

3.5. Discussion

Due to the difficult structure of the problem, we have been unable to prove convergence or optimality of LRGP using existing techniques [2]. However, despite its heuristic nature, LRGP has shown good convergence and optimality characteristics. We discuss these results in detail in Section 4.

We have described LRGP as a distributed algorithm with synchronous stages. For example, consumer allocation and price computation in the current iteration are dependent on rate allocation. It is interesting to consider whether there is any advantage to a centralized, non-distributed formulation; and, to what extent it is possible to relax synchrony requirements between iterations.

LRGP can be trivially centralized by colocating all the different algorithmic pieces in one node. This centralization saves the communication overhead associated with each iteration at the expense of updates required when changes occur to the constants defining the problem, such as n_j^{max} , resource capacities and costs. However, this centralization does not simplify the overall optimization problem. Unlike problems with a concave objective function and convex constraint set, which have straightforward optimizations in a centralized setting (e.g., conditional gradient methods), our problem does not seem amenable to an obvious simplification in a centralized setting.

Our synchronous formulation of LRGP can be made asynchronous using known techniques. For instance, in the flow control algorithm presented by Low *et al.* [23], the authors average over the last few prices from a resource, and

class	flow	nodes	n^{max}	rank
0,1	0	S0,S2	400	20
2,3	0	S0,S2	800	5
4,5	0	S0,S2	2000	1
6,7	1	S0,S1	1000	15
8,9	2	S1,S2	1500	10
10,11	3	S0,S2	400	30
12,13	3	S0,S2	800	3
14,15	3	S0,S2	2000	2
16,17	4	S0,S1	1000	40
18,19	5	S1,S2	1500	100

Table 1. Base Workload

average the last few rates from a source, to allow for missing prices or rates. A similar approach could be used to average computed flow rates, consumer allocations, or prices. However, it is not clear that an asynchronous formulation is beneficial since LRGP is not intended for “live” flow control (e.g., to prevent congestion collapse).

4. Results

In this section, we evaluate our optimization algorithm through simulation. We have two qualitative goals: to evaluate convergence characteristics as a function of workload scaling and variations in utility function; and, to evaluate the quality of the overall utility generated by the optimizer. We evaluate convergence by measuring algorithm performance over several workloads. We examine the quality of the result by comparing total utility to the utility computed by a centralized simulated annealing solution on the same workload. We have experimented with both synchronous and asynchronous versions of LRGP. However, due to space constraints we only present results for the synchronous version. We refer the reader to [24] for more experiments involving the asynchronous LRGP.

4.1. Workload

Lacking any benchmark workloads for utility-based event infrastructures, we have constructed several test workloads by specifying a set of flows, rate limits, consumer classes, class populations, class utilities, node and link capacities, and resource coefficients. To measure scalability we scale the test workload in a controlled fashion. Likewise, we measure sensitivity to utility function by considering several parameterizations of an alternative utility function.

Our test workload has no link bottlenecks, since the focus is on evaluating how node pricing influences both rate and admission control³. In the absence of link bottlenecks, we can ignore the exact topology of the network, and describe the workload only in terms of the flows and the consumer hosting nodes they are routed to.

³Link pricing for rate control has been described in previous work, such as Low *et al.* [23]

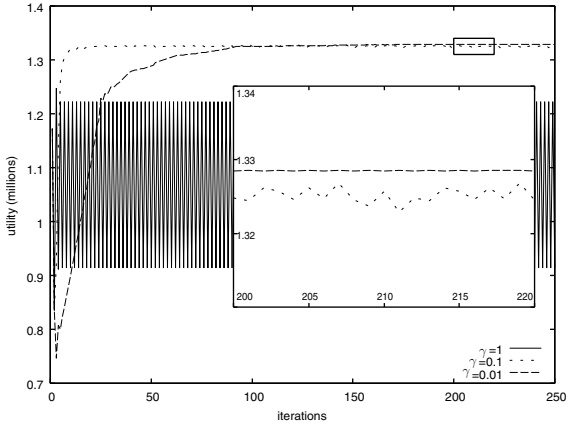


Figure 1. The effect of damping

The basic test workload has six flows numbered $0, \dots, 5$, and three nodes hosting consumers (called *consumer nodes* or *c-nodes* for short) labeled S0, S1 and S2. There are a total of 20 consumer classes, and pairs of consumer classes are identical in utility functions, flow and n^{max} , and differ only in the node they connect to. The utility function for class j which consumes flow i is of the form $rank_j \times f(r_i)$, where the function f is the same across all the flows and r_i is the rate of flow i . The rank can be considered the importance of the class. We experiment with four different $f(r)$: $\log(1+r)$, $r^{0.25}$, $r^{0.5}$, $r^{0.75}$.

Table 1 gives the parameterization of each class. When multiple classes consume the same flow, the number of consumers goes up as the rank decreases. This models different categories of users with less important users being more numerous. The resource model is the same across all flows, consumer classes, and nodes with $F_{b,i} = 3$, $G_{b,j} = 19$ and $c_b = 9 \times 10^5$. These values were measured on the Gryphon publish/subscribe system [1]. Each flow is routed only to the nodes where its consumer classes are present, and all flows have the same $r^{min} = 10$ and $r^{max} = 1000$.

4.2. Convergence

The first focus is on the convergence properties of algorithm. We choose the function $rank_j \times \log(1+r_i)$ to compute the utility function for each class and we run the simulation three times, each time stopping it after 250 iterations. An iteration consists of a rate allocation run by all flow sources and a consumer allocation executed at each consumer node. We measure the value of the global system utility after each step. Throughout the experiments, we assume that the stepsizes used in the node price adjustment formula, γ_1 and γ_2 , are equal to each other. We refer to them simply as γ . Figure 1 depicts the system utility for three different values of γ : 1, 0.1 and 0.01.

When there is no damping ($\gamma = 1$), the utility oscillates with a large amplitude. If damping is used ($\gamma = 0.1$ or $\gamma = 0.01$), the utility stabilizes to a low-amplitude shape.

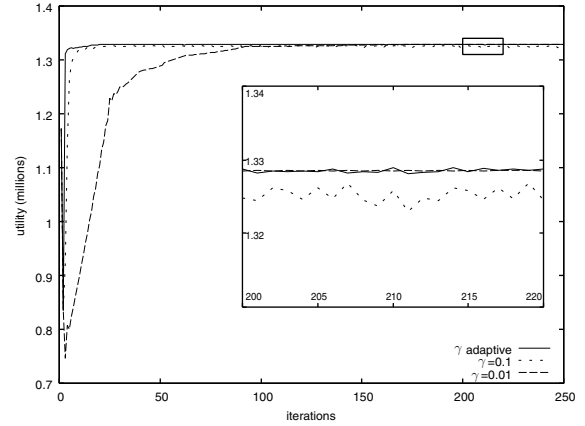


Figure 2. The effect of adaptive γ

The number of iterations needed to achieve stability depends on the value of γ . As shown in the same figure, when $\gamma = 0.1$, the large fluctuations stop after less than 10 iterations, while for $\gamma = 0.01$ the equilibrium occurs after almost 100 iterations. This indicates that larger values for γ lead to faster convergence. Nevertheless, there is a tradeoff involved here, since making γ large also makes the fluctuations larger. We enhance a small rectangular portion of Figure 1 and display it as an inset to reveal this behavior.

As such, we should start with a large γ to ensure fast stabilization, then we should decrease γ to minimize the size of the fluctuations. We have implemented the following heuristic, based on experimentation, to adaptively change the value of γ :

1. start with a fixed value of γ (e.g., 1)
2. as long as the price does not fluctuate, incrementally increase γ by 0.001 at each iteration
3. when fluctuations are detected, decrease γ by half at each iteration

At first, we bounded γ to the interval $[0, \infty)$ (Equation 12). However, considering the discussion above, a large value for γ increases the fluctuations and a low value delays the convergence. Therefore, we constrain γ to a smaller interval, $[0.001, 0.1]$.

We compare the results for adaptive γ to those for fixed γ in Figure 2. The utility converges faster when γ is adaptive. In the inset, we enlarge the rectangular part between iterations 200 and 220. The graph shows that the adaptive γ also induced small fluctuations in the value of the utility.

A direct implication of using an adaptive value for γ is the ability of the algorithm to quickly recover when a flow source leaves the system. Although we do not expect a source to leave often, we would like to minimize the effects on the other nodes when it does happen. We present the variation of utility after we remove flow 5 in Figure 3. We choose flow 5 because it serves consumers from the highest ranked class and thus its departure has the highest effect on the utility. We present the shapes of the utility only between

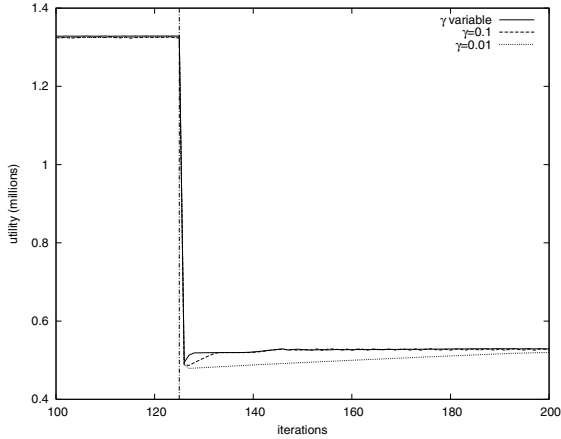


Figure 3. The effect of adaptive γ on recovery from system changes

iterations 100 and 200 and we mark the moment flow 5 is removed by a vertical line. The utility recovers much quicker and stabilizes to low fluctuations when γ is adaptive. From now on, if it is not specified otherwise, all the results will be presented for an adaptive γ .

4.3. Scalability

We look next at the convergence properties and the quality of the results as we scale the number of consumer nodes and the number of flows.

We start with the base workload described above and scale it in two ways:

1. First, we model how the same amount of information propagates to more consumers. Specifically, we increase the number of consumer nodes, while maintaining the number of flows constant. The new consumer nodes have the same characteristics as the initial ones.
2. Second, we model how the system accommodates new information flows. Since each new flow may serve new consumers, we also increase the number of consumer nodes.

We ran our algorithm on the scaled workloads and present the results in Table 2. Ignore the columns associated to Simulated Annealing for now. The first column associated to LRGP shows the number of iterations needed until convergence. The time to complete an iteration equals approximately the maximum round trip time between any two nodes in the overlay. We consider that convergence has occurred when the amplitude of the oscillations in utility becomes less than 0.1% of the value of the utility. As it can be seen, all the tested workloads have similar numbers of iterations until convergence. The LRGP utility column shows the values of the utility immediately after convergence for each of the workloads. These values are growing linearly with the number of consumer nodes in the system. We conclude that neither the convergence properties nor the quality of re-

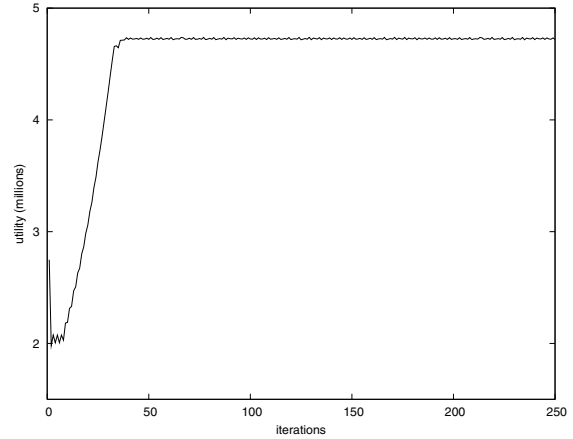


Figure 4. Global utility when the class utility is $rank_j \times r^{0.75}$, for each class j

sults of our algorithm suffer from increasing the number of consumer nodes and the number of flows.

4.4. Optimality

As described in Section 5, the nonconcavity of our utility functions and the nonconvex solution space eliminate related algorithms which are capable of finding optimal solutions. Likewise, the size of the solution space does not allow exhaustive search for the workloads we have presented here. Therefore, to test the quality of our solution, we evaluated the same workloads using a centralized approach based on simulated annealing [17]. Simulated annealing (SA) is a well-known random search technique for finding a global optimum in a large search space. Compared to other random search techniques, simulated annealing can be particularly effective due to the ability to take “backward steps” and therefore escape local maxima. Due to space limitations, we only describe the “cooling schedule” used to evaluate our workloads.

Our cooling schedule consists of a start temperature and a limit on the total number of steps in the simulation. Choosing an appropriate starting temperature depends on the problem structure. A solution space with many local maxima will require more backtracking (*i.e.*, higher start temperature) to increase the probability of finding the global maximum. Since it’s not clear how much backtracking is needed, we evaluated each workload over a range of four start temperatures: 5, 10, 50, 100. We decrease temperature by multiplying the current temperature by 0.999 at the end of each simulation round. Simulation ends when temperature is less than or equal to 1. Likewise, we used three different limits on the total number of steps in each simulation: 10^6 , 10^7 , or 10^8 . Total steps were equally divided among each annealing temperature.

With four temperature ranges and three step limits, each workload was simulated twelve times. Table 2 reports the best results from each simulated annealing run, with run-

Workload	Simulated Annealing				LRGP		Utility Increase
	Start Temp	Max Steps	Runtime (minutes)	Utility	Iterations until convergence	Utility	
6 flows, 3 c-nodes	5	10^8	23	1,248,063	21	1,328,821	6.47%
12 flows, 6 c-nodes	10	10^8	80	2,462,047	21	2,657,600	7.94%
24 flows, 12 c-nodes	5	10^8	357	4,591,584	24	5,313,612	15.72%
6 flows, 6 c-nodes	10	10^8	39	2,473,863	22	2,656,706	7.39%
6 flows, 12 c-nodes	5	10^8	65	4,723,709	22	5,313,412	12.48%
6 flows, 24 c-nodes	10	10^8	148	8,948,644	22	10,626,824	18.75%

Table 2. Quality of results for LRGP and Simulated Annealing as the size of the system grows

Utility function	Simulated Annealing				LRGP		Utility Increase
	Start Temp	Max Steps	Runtime (minutes)	Utility	Iterations until convergence	Utility	
$rank_j \times \log(1+r)$	5	10^8	23	1,248,063	21	1,328,821	6.47%
$rank_j \times r^{0.25}$	50	10^8	28	876,068	23	926,185	5.72%
$rank_j \times r^{0.5}$	50	10^8	31	1,989,301	28	2,003,225	0.69%
$rank_j \times r^{0.75}$	50	10^8	31	4,677,350	39	4,735,044	1.23%

Table 3. Convergence and quality of results as the utility function of a class varies

time specified in minutes. We also compare total utility with the utility achieved by our optimization algorithm. In each case, LRGP was able to find a better utility, in the best case showing an 18.75% increase. As a general trend, simulated annealing showed poorer performance as the number of independent variables increased (*e.g.*, compare 6 flows, 3 consumer nodes to 12 flows, 6 consumer nodes, which roughly doubles the number of independent variables).

4.5. Changing the Shape of the Utility

We have presented experimental results using a logarithmic utility function for each class j , of the form $rank_j \times \log(1+r)$. We analyze now other shapes of the utility, specifically those given by $rank_j \times r^{(k)}$, where k is 0.25, 0.5, or 0.75. Table 3 shows the results for the base workload. The number of iterations until convergence increases as the value of the exponent increases towards 1. The explanation for this is that as k increases, the slope of the utility function also increases. This causes a small variation in price to translate into a progressively larger variation in rate, which in turn produces slower convergence.

To evaluate optimality, we report simulated annealing results using the modified workloads. As in the base workloads, our algorithm achieved consistently better results.

5. Related Work

The research related to the distributed optimization algorithm we have proposed spans a wide range of topics: network optimization for unicast and multicast flows, admission control, path optimization in networks, and stream based overlays.

Several approaches for maximizing the system utility have been proposed in the area of network flow optimization. Due to practical reasons, simple and distributed al-

gorithms are desirable and one of the most common techniques to achieve this is the dual decomposition [2, 25]. Relevant research includes work in both unicast flow optimization [23, 16, 20, 21] and multicast flow optimization [15, 29]. Our algorithm also uses the idea of dual problem decomposition but unlike the aforementioned work, the constraints are nonconvex because we consider node constraints. Moreover, network flow optimization is based only on flow rates as the variables that determine the maximum system utility. In contrast, we explicitly consider admission control so that the objective function of the optimization also depends on the number of consumers for each flow.

Multicast flow control algorithms [15, 29] consider multirate flows but do not account for the resources consumed to modify the rate from one link to the other. Since the change of rate happens at nodes, this has practically no effect on link constraints. However, if node resources were also considered, as we do in our optimization, the problem would become harder. We defer the study of multirate allocation for future work.

Recall that our initial optimization problem is nonconvex. A similar problem space has been considered in the context of rate control in networks [22, 9, 6]. Just as in the convex network flow optimization, however, the objective function in these approaches depends only on rates and the constraint sets are convex.

Admission control has been used in networks to ensure that admitting a new flow still guarantees the commitments (*e.g.*, bounded delay, guaranteed rate) made by the network to the already admitted flows. Two basic approaches to admission control have been proposed: (1) a parameter-based approach [10, 7], which uses *a priori* characterizations of sources to calculate the worst-case behavior of all existing flows; and (2) a measurement-based approach [13, 14, 11],

which relies on traffic measurements to estimate the behavior of a flow. Our admission control scheme is closer to the latter, since it relies on the current rates of the flows to make admission decisions. However, we admit consumers, not flows. Furthermore, we do not guarantee any commitments made by the system to the already admitted consumers; we are merely trying to regulate the number of consumers such that we obtain an increase in the global system utility. That is why consumers that bring less benefit can be unadmitted.

We consider the problem of path optimization and load distribution as discussed in the networking literature [8, 12] or in the context of stream based overlay networks [26, 28, 18, 19, 30] complementary to our work. Our optimization algorithm assumes all the flows have a given path and, since flows suffer transformations on their path, it is not recommended to change paths very frequently. We believe, however, that occasional change of paths may improve the overall load and therefore is beneficial.

6. Conclusion

Our primary contribution is a distributed algorithm for optimizing global utility in an event-driven infrastructure. Our algorithm is efficient and scalable in an environment where both rate and admission control are considered. The key insight of our approach involves partitioning the optimization problem into two subproblems: a greedy approach which controls the admission of consumers and a Lagrangian approach which controls the rate allocation for each flow. We use link and node prices to make tradeoffs between admission control and rate control.

Experimental results show that our algorithm exhibits good convergence properties and scales well with the number of flows and the number of consumers. Likewise, the solution generated by our approach was superior (in terms of total utility and time) when compared to a simulated annealing approach. Thus, although our results may not be optimal, we have some confidence that they provide an effective solution for many systems.

Acknowledgements

We thank Bobby Bhattacharjee, Neil Spring and the anonymous reviewers for their insightful comments.

References

- [1] <http://www.research.ibm.com/distributedmessaging>.
- [2] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.
- [3] S. Bholra, M. Astley, R. Saccone, and M. Ward. Utility-aware resource allocation in an event processing system. In *3rd IEEE Conference on Autonomic Computing (ICAC)*, 2006.
- [4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM TOCS*, 19(3):332–383, 2001.
- [5] D. A. Chappell. *Enterprise Service Bus*. O’Reilly, 2004.
- [6] M. Chiang, S. Zhang, and P. Hande. Distributed rate allocation for inelastic flows: Optimization framework, optimality conditions, and optimal algorithms. In *IEEE Infocom*, 2005.
- [7] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in integrated services packet network: Architecture and mechanism. In *ACM SIGCOMM*, 1992.
- [8] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *IEEE Infocom*, 2001.
- [9] M. Fazel and M. Chiang. Network utility maximization with nonconcave utilities using sum-of-squares method. In *IEEE Control and Decision Conference*, 2005.
- [10] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE JSAC*, 8(3):368–379, 1990.
- [11] R. J. Gibbens and F. P. Kelly. Measurement-based connection admission control. In *15th Int’l Teletraffic Congress*, 1997.
- [12] T. Guven, C. Kommareddy, R. J. La, M. A. Shayman, and B. Bhattacharjee. Measurement based optimal multi-path routing. In *IEEE Infocom*, 2004.
- [13] J. Hyman, A. A. Lazar, and G. Pacifici. A separation principle between scheduling and admission control for broadband switching. *IEEE JSAC*, 11(4), 1993.
- [14] S. Jamin, P. B. Danzig, S. J. Shenker, and L. Zhang. A measurement-based admission control algorithm for integrated service packet networks. *IEEE/ACM ToN*, 5(1), 1997.
- [15] K. Kar, S. Sarkar, and L. Tassiulas. Optimization based rate control for multirate multicast sessions. In *IEEE Infocom*, 2001.
- [16] F. P. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, 1998.
- [17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), 1983.
- [18] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-aware distributed stream management using dynamic overlays. In *IEEE ICDCS*, 2005.
- [19] V. Kumar, B. F. Cooper, and K. Schwan. Distributed stream management using utility-driven self-adaptive middleware. In *ICAC*, 2005.
- [20] S. Kunniyur and R. Srikant. End-to-end congestion control schemes: Utility functions, random losses and ECN marks. *IEEE/ACM ToN*, 11(5):689–702, 2003.
- [21] R. J. La and V. Anantharam. Charge-sensitive TCP and rate control in the Internet. In *IEEE Infocom*, 2000.
- [22] J.-W. Lee, R. R. Mazumdar, and N. B. Shroff. Non-convex optimization and rate control for multi-class services in the Internet. In *IEEE Infocom*, 2004.
- [23] S. H. Low and D. E. Lapsley. Optimization flow control I: Basic algorithm and convergence. *IEEE/ACM ToN*, 7(6):861–874, 1999.
- [24] C. Lumezanu, S. Bholra, and M. Astley. Utility optimization for event-driven distributed infrastructures. IBM Research Report RC 23916, 2006.
- [25] D. P. Palomar and M. Chiang. On alternative decompositions and distributed algorithms for network utility problems. In *IEEE Globecom*, 2005.
- [26] P. Pietzuch, J. Shneidman, M. Welsh, M. Seltzer, and M. Rousopoulos. Path optimization in stream-based overlay networks. Technical report, Harvard University, 2004.
- [27] S. Shenker. Fundamental design issues of the future Internet. *IEEE Journal on Selected Areas in Communications*, 1995.
- [28] J. Shneidman, P. Pietzuch, M. Welsh, M. Seltzer, and M. Rousopoulos. A cost-space approach to distributed query optimization in stream based overlays. In *NetDB*, 2005.
- [29] W.-H. Wang, M. Palaniswami, and S. H. Low. Necessary and sufficient conditions for optimal flow control in multirate multicast networks. *IEE Proceedings-Communications*, 2003.
- [30] Y. Xing, S. Zdonik, and J.-H. Hwang. Dynamic load distribution in the Borealis stream processor. In *ICDE*, 2005.