Ann Oper Res (2009) 170: 161–182
DOI 10.1007/s10479-008-0434-y

A decentralized control mechanism for stream processing networks

Zhen Liu · Ao Tang · Cathy H. Xia · Li Zhang

Published online: 17 September 2008

Abstract Data streaming applications are becoming more and more common due to the rapid development in emerging areas such as sensor networks, multimedia streaming, and on-line data mining, etc. These applications are often running in a decentralized, distributed environment. The requirements for processing large volumes of streaming data at real time have posed many great design challenges. One of the critical issues is to optimize the on-going resource consumption of multiple, distributed, cooperating processing units. In this paper, we consider a generic model for the general stream data processing systems. We address the resource allocation problem for a collection of processing units so as to maximize the weighted sum of the throughput of different streams. Each processing unit may require multiple input data streams *simultaneously* and produce one or many valuable output streams. We develop decentralized control mechanisms that maximize the overall system throughput in such data stream processing networks. Performance analysis on the optimality and complexity of these mechanisms are also provided.

Keywords Stream processing · Resource allocation · Distributed algorithm

1 Introduction

The rapid development of the network technologies has triggered the emergence of many new applications. Stream data processing is one of the most interesting and challenging ap-

L. Liu · C.H. Xia (✉) · L. Zhang
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA
e-mail: cathyx@us.ibm.com

L. Liu
e-mail: zhenl@us.ibm.com

L. Zhang
e-mail: zhangli@us.ibm.com

T. Tang
Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125, USA
e-mail: aotang@caltech.edu

plications that are under extensive study by the research community. In such applications, continuous data streams arriving to the system need to be processed by multiple processing units in real-time to generate streams of desirable results. One example of this type of application is network monitoring and management. Continuous streams of network usage information are collected from various monitoring points in the network. These data need to be analyzed and correlated on the fly to determine whether the network is in a normal running mode, or is under intrusive attacks. Many financial applications such as the stock quote and trading systems also exhibit this type of characteristics. Continuous quote and trade data streams need to be processed in real-time. Another area where many stream data processing applications arise is in the context of sensor networks. The requirements to process a large volume of data at real time for these continuous processing applications have posed many great design challenges.

Stream data processing systems typically require a large amount of processing power with many different computers in order to achieve satisfactory performance levels. Multiple processing units often share a pool of computing resources. One important problem is to find the best resource allocation scheme for the multiple processing units to efficiently utilize the available resources. As in most real time systems, applications are often running in a decentralized environment. The resource allocation scheme also has to be decentralized in nature.

This paper addresses the fundamental resource allocation problem raised in the above setting. We consider a generic stream data processing model, in which there are multiple data streams, each passing through a sequence of processing units to generate useful output streams. The processing units serving different data streams share the same underlying computing resources. Sub-optimal allocation of the resources may lead to under-utilizing certain processing units and over-utilizing some others. Our goal is to obtain a decentralized control mechanism that maximizes the weighted sum of the throughput of different output streams. In our model, each processing unit may require multiple input data streams *simultaneously* and produce one or many valuable output streams. Such kind of simultaneous flow consumption is related to the fork-join mechanism in queueing applications (Baccelli and Liu 1989, 1990) and also arises in the context of supply chain management (Baldwin et al. 2000; Simchi-Levi et al. 2002; Mas-Collel et al. 1995). It is an important feature in many streaming processing applications. For example, the network usage information from multiple routers needs to be correlated to derive the overall user flow information. Another distinct characteristic in our model is the introduction of the shrink/expansion factor for the flows at each processing units. The volume of the output data stream can be different from the volume of the input data stream at each processing unit. Such a phenomenon naturally occurs in the join, filter and selection mechanisms in streaming applications (Viglas and Naughton 2002).

In this paper, we present an analytical approach to solve the generic stream data processing problem. We first develop the optimal solution for several special cases, including the case with a single output and the case with a tree topology. For the single output case, we propose a backward algorithm which produces an optimal solution in linear time. For the tree case, we provide a backward shrink algorithm which also yields an optimal solution in linear time. Based on the algorithm for trees, we propose two distributed algorithms to find the best, or close to optimal solutions in a general network with multiple streams. The algorithms are based on an aggregation heuristic that aggregates local subgraphs into equivalent super nodes, where the super nodes can play the role as a cluster head or local manager. We present experimental results to demonstrate the quality of our distributed solutions.

The paper is organized as follows. Section 2 presents the general model. The centralized resource allocation problem under static settings is formulated in Sect. 3. We then investigate the structural properties of the optimal solutions for special cases (a single output stream case and the tree case) in Sects. 4 and 5. Section 6 presents general decomposition rules for special graphs including the series-parallel graphs, and graphs that can be partitioned by a single link. In Sect. 7, we propose two distributed solutions for the general resource allocation problem based on the optimal algorithms derived previously. Experimental evaluations of the effectiveness of these solutions are also presented. Concluding remarks are provided in Sect. 8.

2 Model

In a stream data processing system, incoming data flows continuously from several sources. These data needs to go through several levels of processing, such as selection, filtering, or combining, to generate the expected output. We use a directed acyclic graph (DAG), referred to as *stream processing graph*, to describe the producer-consumer relationship among processing units associated with the streams. There are source nodes, sink nodes and processing nodes in the graph, with directed edges represent the information flow between various nodes. The source nodes correspond to the source of the input data streams. These nodes only have edges going out, and do not have any edges between them. The sink nodes correspond to the receivers of the eventual processed information. These nodes only have edges going to them, and do not have any edges in between. Processing nodes stand for processing units. A processing unit may require inputs from multiple data streams simultaneously and produce one or many valuable output streams. Such a graph can be plotted in a way such that all the directed edges are pointing downward. We can now view the system as information coming from the top and passing through the processing units in the middle and eventually leading to the output streams at the bottom, see Fig. 1.

Denote $\mathcal{I}, \mathcal{P}, \mathcal{O}$ respectively the set of source, processing, and sink nodes in the graph, as illustrated in Fig. 1. Let $\mathcal{E}$ denote the set of all the directed edges. Denote the underlying graph by $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \mathcal{I} \cup \mathcal{P} \cup \mathcal{O}$. We assume G is connected. For each $j \in \mathcal{N}$, let $\mathcal{I}^j$ denote the set of immediate predecessors, i.e., all nodes i such that the directed edge (i, j) is in $\mathcal{E}$.

Let $\mathcal{O}^j$ denote the set of immediate successors, i.e., all the nodes k such that the directed edge (j, k) is in $\mathcal{E}$. Without loss of generality, we assume that each source node produces a single stream as the input to the processing nodes, and there is exactly one output stream leading to each sink node. Therefore, $|\mathcal{O}^i| = 1$ for all source nodes $i \in \mathcal{I}$, and $|\mathcal{I}^k| = 1$ for all sink nodes $k \in \mathcal{O}$.

We now describe the quantitative relationship between the input, output and resource consumption. In our model, each processing unit processes multiple data flows from its upstream nodes at a given proportion in order to generate output flows to its downstream nodes at a possibly different proportion. Each processing unit $j \in \mathcal{P}$, with a unit of CPU resource, will process α_i^j amount of flow from node i for all $i \in \mathcal{I}^j$, and generate β_k^j amount of flow to node k for all $k \in \mathcal{O}^j$. Here, the superscript j always represents the current node under consideration. For all the source nodes $j \in \mathcal{I}$, let λ_j be their flow input rates, where $0 < \lambda_j \leq \infty$. Each unit of output flow to node $k \in \mathcal{O}$ has value w_k.

We assume all the parameters λ, α, β and w are positive, as is the case in most real applications. In general, quantities α_i^j and β_k^j, although measurable, are not deterministic. They typically depend on the input data. Throughout this paper, unless specifically stated, we

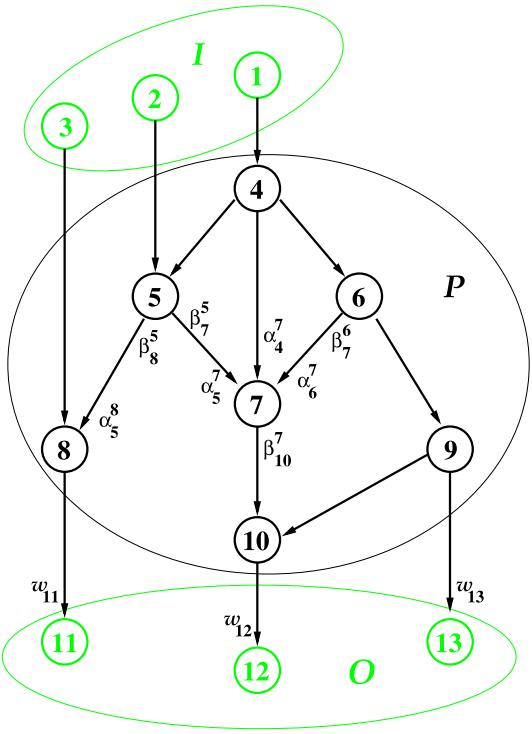


Fig. 1 Graph representation of the problem

shall assume that this dependence is stationary. The quantities α_i^j and β_k^j are defined as the average consumption and production rates, respectively. The case of changing consumption and production rates will be discussed in Sect. 7.

Assume we have a total of R units of CPU resource available. Our goal is to find optimal or approximate solutions for allocating the resource among all the processing units to maximize the weighted sum (e.g., based on the importance) of the throughput of the output streams. We are interested in finding distributed solutions capable of adapting to local changes in the consumption and production rates.

3 Static problem with global information

Assume that the consumption and production rates (α_i^j and β_k^j) are known constants. Then, the general problem stated above can be formulated as a linear program (LP) problem. Let's consider a resource allocation policy $\mathbf{x} = (x_j,\ j \in \mathcal{P})$, where x_j is the amount of resource assigned for processing unit j, $j \in \mathcal{P}$. Node j, with x_j amount of resource, can then process $\alpha_i^j x_j$ amount of flow for each node $i \in \mathcal{I}^j$, and generate $\beta_k^j x_j$ amount of flow for each node $k \in \mathcal{O}^j$. For such an allocation to be valid, the amount of flow consumed by the downstream nodes cannot be larger than the amount generated by the upstream nodes. And this needs to hold for all the edges in the graph, including the input edges. The overall benefit from the output streams is $\sum_{k\in\mathcal{O}} \sum_{j\in\mathcal{I}^k} w_k \beta_k^j x_j$. The optimization problem can be formulated as the

following linear program.

$$
\begin{aligned}
\text{(LP)} \quad \max \quad & \sum_{k\in\mathcal{O}} \sum_{j\in\mathcal{I}^k} w_k \beta_k^j x_j := V(\mathbf{x}) \\
\text{s.t.} \quad & \sum_{j\in\mathcal{P}} x_j \le R, \quad \text{(capacity constraint)}, \\
& \alpha_i^j x_j \le \lambda_i, \quad i \in \mathcal{I}, j \in \mathcal{O}^i \quad \text{(input flow constraints)}, \\
& \alpha_i^j x_j \le \beta_j^i x_i, \quad j \in \mathcal{O}^i \cap \mathcal{P}, i \in \mathcal{P} \quad \text{(internal flow constraints)}, \\
& x_j \ge 0, \quad j \in \mathcal{P}.
\end{aligned}
$$

One can use general linear program solvers to solve the (LP) problem once the parameters are available. However, this centralized solution requires global information and needs all the processing nodes to act in coordination. It is not dynamic enough to adapt to the possible changing environment, where certain flow properties and system parameters could vary over time. Distributed algorithms however, require local information, and are more adaptive to the changing environment. We will develop distributed algorithms for this resource allocation problem based on the properties from the global optimal solutions. For this purpose, we next derive some key properties of optimal solutions.

For the ease of presentation, we define a partial order of the nodes in graph $G = (\mathcal{N}, \mathcal{E})$. For two nodes i and j in $\mathcal{N}$, we say that $i \prec j$ if there is a directed path $(i = l_1, l_2, \ldots, l_L = j)$ that leads node i to node j, where $(l_j, l_{j+1}) \in \mathcal{E}$ for $j = 1, \ldots, L-1$.

We shall say that a node is active (resp. idle) in a solution if this node receives nonzero (resp. zero) CPU resource allocation in that solution. The following lemma is immediate.

Lemma 1 *If node $i \in \mathcal{P}$ is active (i.e., non-idle), then all nodes $k \in \mathcal{P}$ with $k \prec i$ must also be active.*

Consider the special case when the source nodes are not rate limited, that is, $\lambda_i = \infty$ for all $i \in \mathcal{I}$. The next two lemmas show some basic properties of the optimal solution to the above (LP).

Lemma 2 *Suppose $\lambda_i = \infty$ for all $i \in \mathcal{I}$. Then, the optimal solution $(x_j^*, j \in \mathcal{P})$ must satisfy $\sum_{j\in\mathcal{P}} x_j^* = R$.*

Proof We prove the result by contradiction. Suppose $\sum_{j\in\mathcal{P}} x_j^* < R$. Define $x_j' = \frac{x_j^* R}{\sum_{k\in\mathcal{P}} x_k^*}$, $j \in \mathcal{P}$. It is easy to see that x' satisfies all the constraints in the (LP). Furthermore, from the assumption, we have $x_j' > x_j^*$ for all $j \in \mathcal{P}$. Therefore, x' is a better solution than x^*. This completes the proof. □

Lemma 2 indicates that the optimal solution to the above (LP) will satisfy the capacity constraint as equality. In other words, the best solution is to use up all the available resources.

Lemma 3 *Suppose $\lambda_i = \infty$ for all $i \in \mathcal{I}$. Denote* LP(R) *the linear program with total resource R, and $V^*(R)$ the corresponding optimal objective value. If $\mathbf{x}^*(R)$ solves* LP(R), *then $\gamma \mathbf{x}^*(R)$ solves* LP(γR), *for all $\gamma \ge 0$. In addition, $\mathbf{V}^*(\gamma R) = \gamma \mathbf{V}^*(R)$.*

Proof Let $y_j = \gamma x_j$, $j \in \mathcal{P}$. By changing variables x_j's to y_j's, the linear program LP(R) is easily checked to be equivalent to a new linear program with R replaced by γR, and the objective function $V(R)$ replaced by $\gamma V(R)$. The claims thus follow immediately. □

Lemma 3 shows that the optimal solution is linear in the total available resource R. The following corollary is immediate from Lemmas 2 and 3.

Corollary 1 *Suppose* $\lambda_i = \infty$ *for all* $i \in \mathcal{I}$. *In order to solve* (LP), *it suffices to consider solutions* $\mathbf{x}$ *which satisfy capacity constraint* $\sum_{j\in\mathcal{P}} x_j = 1$. *Then* $\mathbf{x}^* := R\mathbf{x}$ *is an optimal solution to* (LP).

4 The single output case

In this section, we consider the case when there is only one final output stream of interest. In other words, $\mathcal{O} = \{O\}$ is a singleton, where O is the only sink node. Without loss of generality, denote node N to be the *last processing node* reaching O (since there is exactly one edge leading to each sink node). In this case, we can have a simple backward algorithm to solve the problem.

Algorithm 1 (Graphs with single output)

1. Initialize: set $A = \{N\}$, and let $x_N = 1$, and $x_i = 0$ for all $i \neq N$.
2. Let $B := \bigcup_{i\in A} \mathcal{I}^i$ be the set of all predecessors of nodes in A.

   - If $B \subset \mathcal{I}$, i.e., all nodes in B are source nodes, then exit;
   - Else, for each $i \in B$, let

$$x_i = x_i \vee \max_{\{j\in A:(i,j)\in\mathcal{E}\}} \frac{\alpha_i^j x_j}{\beta_j^i}. \tag{1}$$

   Set $A = B$ and go back to step 2.

Remark 1 Since each link in the graph is checked exactly once, the total complexity of Algorithm 1 is $O(|\mathcal{E}|)$.

Remark 2 Algorithm 1 can be easily implemented in a distributed manner since it only requires information exchange between neighbors. At each iteration, every node passes its current x value and the α and β parameters to its immediate predecessors, and then updates the x value based on (1). Clearly, after $O(|\mathcal{E}|)$ iterations, the resulting x value would be the same as the one produced by Algorithm 1.

Lemma 4 $\mathbf{x} = (x_j, j \in \mathcal{P})$, *the allocation produced by Algorithm 1, gives the minimum resource requirement at each node* $j \in \mathcal{P}$, *and the minimum input rate requirement from each source node in* $\mathcal{I}$ *in order for node* O *to receive output at rate* β_O^N.

Proof In order for node O to receive output at rate β_O^N, node N must produce output stream at rate β_O^N. Node N requires at least 1 unit of resource and inputs at rate $\alpha_i^N x_N$ from all its predecessor nodes $i \in B$. Here $A = \{N\}$ and $B = \mathcal{I}^N$.

For each node $i \in B$, in order to provide output to node N at requested rate $\alpha_i^N x_N$, the minimum amount of resource required at node i is $x_i = \frac{\alpha_i^N x_N}{\beta_N^i}$; furthermore, node i requires inputs at rate α_j^i from all its predecessor nodes. If node i is the parent of both nodes j and j' ($j, j' \in A$), then i needs to meet the input requirements of both j and j'. Hence, the minimum amount of resource required at node i is: $x_i = x_i \vee \max_{\{j \in A:(i,j)\in\mathcal{E}\}} \frac{\alpha_i^j x_j}{\beta_j^i}$.

Repeat the above argument until we reach the top of the graph, where all nodes are source nodes. The resulting resource requirement at each node in $\mathcal{P}$, and the input rates from all source nodes in $\mathcal{I}$ are the minimum requirements in order for sink node O to receive output at rate β_O^N. □

In fact, Algorithm 1 can be applied to any subgraph G_1 which has a single end node N. The resulting allocation $\mathbf{x}$ provides the minimum resource requirements for all nodes in G_1 and minimum input rate requirements for all input links into G_1 in order to produce output at rate β_Q^N for all $Q \in \mathcal{O}^N$.

Theorem 1 *Let* $\mathbf{x} = (x_j, j \in \mathcal{P})$ *be the allocation produced by Algorithm 1. Then, for a system with total resource* R *and maximum input rates* $\lambda_i, i \in \mathcal{I}$, *the optimal resource allocation* $\mathbf{x}^*$ *is given by*

$$x_i^* = \min\left(\delta_{\max}, \frac{R}{\sum_i x_i}\right) \cdot x_i, \quad i \in \mathcal{P}, \tag{2}$$

where

$$\delta_{\max} := \max\{\delta > 0 : \delta\alpha_i^j x_j \le \lambda_i, i \in \mathcal{I}, (i,j) \in \mathcal{E}\}. \tag{3}$$

and the total return is $V^* = \min(\delta_{\max}, \frac{R}{\sum_i x_i})\beta_O^N$.

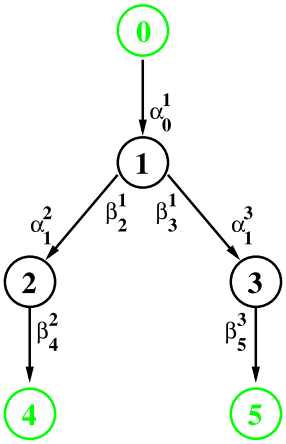
Proof Based on Lemma 4, since $\mathbf{x} = (x_j, j \in \mathcal{P})$ gives the minimum requirement on the amount of resources needed in order to output at rate β_O^N. With total available capacity R, the maximum output rate is bounded above by $\frac{R}{\sum_i x_i}\beta_O^N$. Similarly, for each source node $i \in \mathcal{I}$, since $\alpha_i^j x_j$ is the minimum required input rate to node j, where $(i,j) \in \mathcal{E}$, in order for the sink node to receive output at rate β_O^N, thus with total available rate λ_i, the maximum output rate is bounded above $\delta_{\max}\beta_O^N$, where $\delta_{\max}$ is defined in (3). Hence $\mathbf{x}^*$ given by (2) must be the optimal allocation. □

5 Multiple output streams: trees

In this section, we generalize the algorithms in the previous section to address the cases with multiple output nodes, i.e., $|\mathcal{O}| > 1$. In this setting, we need to decide between generating output for one stream versus generating output for another stream, or both. This kind of trade-off is not obvious to evaluate due to the simultaneous flow consumption and output. We will first derive the algorithms to treat certain simpler cases. And then extend the solution to address the general cases.

We first consider the case where the graph G is a tree. That is, after changing all the directed edges to undirected edges, the resulting graph is a tree. We start with the basic case in the following example in Fig. 2.

Fig. 2 A 3-node binary tree



Example 2 (A binary tree) We now define a linear program as follows,

$$\begin{aligned}
\max \quad & w_4\beta_4^2 x_2 + w_5\beta_5^3 x_3 \\
\text{s.t.} \quad & x_1 + x_2 + x_3 \le R, \\
& \alpha_0^1 x_1 \le \lambda_0, \\
& \alpha_1^2 x_2 \le \beta_2^1 x_1, \\
& \alpha_1^3 x_3 \le \beta_3^1 x_1, \\
& x_1 \ge 0, \qquad x_2 \ge 0, \qquad x_3 \ge 0.
\end{aligned}$$

In the event this maximization problem has more than one optimal solution, we let $(x_1^\#, x_2^\#, x_3^\#)$ be the optimal solution that minimizes the total allocated capacity $x_1 + x_2 + x_3$. We use this convention for the optimal solutions in all the optimization problems considered in this paper.

Theorem 2 *The optimal resource allocation solution for the binary tree system can be one of the following three cases*:

(i) *If* $\frac{w_4\beta_4^2\beta_2^1}{\alpha_1^2+\beta_2^1} > w_5\beta_5^3$, *then the system only produces output* 2, *with no production of output* 3, *with*

$$x_2^\# = \frac{\beta_2^1}{\alpha_1^2+\beta_2^1} r, \qquad x_3^\# = 0, \qquad x_1^\# = \frac{\alpha_1^2}{\alpha_1^2+\beta_2^1} r,$$

where $r = \min(R, \frac{\alpha_1^2+\beta_2^1}{\alpha_0^1\alpha_1^2}\lambda_0)$.

(ii) *If* $\frac{w_5\beta_5^3\beta_3^1}{\alpha_1^3+\beta_3^1} > w_4\beta_4^2$, *then the system only produces output* 3, *with no production of output* 2, *with*

$$x_2^\# = 0, \qquad x_3^\# = \frac{\beta_3^1}{\alpha_1^3+\beta_3^1} r, \qquad x_1^\# = \frac{\alpha_1^3}{\alpha_1^3+\beta_3^1} r,$$

where $r = \min(R, \frac{\alpha_1^3+\beta_3^1}{\alpha_0^1\alpha_1^3}\lambda_0)$.

(iii) *Else, the system produces a weighted sum of both output* 2 *and* 3, *with*

$$x_2^{\#} = \frac{\alpha_1^3\beta_2^1}{\alpha_1^2\alpha_1^3 + \alpha_1^3\beta_2^1 + \alpha_1^2\beta_3^1} r, \qquad x_3^{\#} = \frac{\alpha_1^2\beta_3^1}{\alpha_1^2\alpha_1^3 + \alpha_1^3\beta_2^1 + \alpha_1^2\beta_3^1} r,$$

$$x_1^{\#} = \frac{\alpha_1^2\alpha_1^3}{\alpha_1^2\alpha_1^3 + \alpha_1^3\beta_2^1 + \alpha_1^2\beta_3^1} r,$$

where $r = \min(R, \frac{\alpha_1^2\alpha_1^3+\alpha_1^3\beta_2^1+\alpha_1^2\beta_3^1}{\alpha_0^1\alpha_1^2\alpha_1^3}\lambda_0)$.

Proof Assume $\mathbf{x}^*$ is the optimal solution and denote $V(\mathbf{x}^*)$ the corresponding objective value.

First consider case (i). If $\frac{w_4\beta_4^2\beta_2^1}{\alpha_1^2+\beta_2^1} > w_5\beta_5^3$ and $x_3^* > 0$, consider another solution with $x_3 = 0$, and $x_2 = x_2^* + \frac{\beta_2^1}{\alpha_1^2+\beta_2^1}x_3^*, \quad x_1 = x_1^* + \frac{\alpha_1^2}{\alpha_1^2+\beta_2^1}x_3^*$. It is easy to check the feasibility of the new allocation. The weighted sum of the output $V(\mathbf{x})$ under this allocation is

$$V(\mathbf{x}) = w_4\beta_4^2 x_2 = w_4\beta_4^2 x_2^* + \frac{w_4\beta_4^2\beta_2^1}{\alpha_1^2+\beta_2^1}x_3^* > w_4\beta_4^2 x_2^* + w_5\beta_5^3 x_3^* = V(\mathbf{x}^*).$$

That is, the total benefit under x is higher than that under allocation x^*, which contradicts the assumption that x^* is optimal. Therefore, $x_3^* = 0$, and the expression of x_2^* and x_1^* are easily obtained since constraint (4) has to be tight in order to be optimal.

Case (ii) can be argued the same way as above.

If it is neither case (i) nor case (ii), we argue that $\alpha_1^2 x_2^* = \beta_2^1 x_1^*$ and $\alpha_1^3 x_3^* = \beta_3^1 x_1^*$, i.e., both constraints in the original problem are tight under the optimal allocation. If neither is tight, clearly it can not be the optimal allocation. Without loss of generality, assume $\alpha_1^2 x_2^* < \beta_2^1 x_1^*$. We can always choose a small enough $\epsilon > 0$ and form a new allocation with

$$x_2 = x_2^* + \epsilon(x_3^* + x_1^*) = x_2^* + \epsilon x_3^*\left(\frac{\alpha_1^3 + \beta_3^1}{\beta_3^1}\right), \qquad x_3 = (1-\epsilon)x_3^*, \qquad x_1 = (1-\epsilon)x_1^*.$$

The objective under this new allocation is

$$\begin{aligned} V(\mathbf{x}) &= w_4\beta_4^2 x_2 + w_5\beta_5^3 x_3 \\ &= w_4\beta_4^2 x_2^* + w_5\beta_5^3 x_3^* + \epsilon x_3^*\left(\beta_4^2 w_4 \cdot \frac{\alpha_1^3 + \beta_3^1}{\beta_3^1} - \beta_5^3 w_5\right) \\ &> w_4\beta_4^2 x_2^* + w_5\beta_5^3 x_3^* = V(\mathbf{x}^*), \end{aligned}$$

where the inequality holds because $w_4\beta_4^2 > \frac{w_5\beta_5^3\beta_3^1}{\alpha_1^3+\beta_3^1}$. In short, the objective under x is higher than that under x^*, which contradicts the optimality of x^*. □

Theorem 3 *Following the above theorem, for each of the cases, the problem in Fig.* 2 *is equivalent to the simpler model in Fig.* 3. *The equivalent parameters* α, β *and* w *are given as follows*:

(i) *If* $\frac{w_4\beta_4^2\beta_2^1}{\alpha_1^2+\beta_2^1} > w_5\beta_5^3$, *then*,

$$\hat{\alpha}_0^1 = \alpha_0^1, \qquad \hat{\beta}_2^1 = \beta_2^1, \qquad \hat{\alpha}_1^2 = \alpha_1^2, \qquad \hat{\beta}_3^2 = \beta_4^2, \qquad \hat{w}_3 = w_4.$$

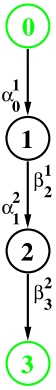


Fig. 3 Equivalent 2-node representation

(ii) *If* $\frac{w_5\beta_5^3\beta_3^1}{\alpha_1^3+\beta_3^1} > w_4\beta_4^2$, *then*,

$$\hat{\alpha}_0^1 = \alpha_0^1, \qquad \hat{\beta}_2^1 = \beta_3^1, \qquad \hat{\alpha}_1^2 = \alpha_1^3, \qquad \hat{\beta}_4^2 = \beta_5^3, \qquad \hat{w}_3 = w_5.$$

(iii) *Else*, *we have*

$$\hat{\alpha}_0^1 = \alpha_0^1, \qquad \hat{\beta}_2^1 = \beta_2^1 + \beta_3^1,$$

$$\hat{\alpha}_1^2 = \frac{\frac{\alpha_1^2}{\beta_2^1}\alpha_1^3 + \frac{\alpha_1^3}{\beta_3^1}\alpha_1^2}{\frac{\alpha_1^2}{\beta_2^1} + \frac{\alpha_1^3}{\beta_3^1}}, \qquad \hat{\beta}_3^2 = \frac{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2 + \frac{\beta_3^1}{\alpha_1^3}\beta_5^3}{\frac{\beta_2^1}{\alpha_1^2} + \frac{\beta_3^1}{\alpha_1^3}}, \quad \text{and}$$

$$\hat{w}_3 = \frac{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2 w_4 + \frac{\beta_3^1}{\alpha_1^3}\beta_5^3 w_5}{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2 + \frac{\beta_3^1}{\alpha_1^3}\beta_5^3}.$$

Notice that these parameter mappings are independent of the parameters λ_0 *and* R. *This is a key property for the later algorithms*.

Proof The proof is straightforward by checking feasibility conditions both ways. It is thus omitted due to space limitation. □

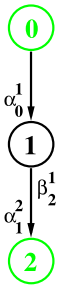
After merging the leaf nodes into a single leaf, we also have another basic reduction to reduce two nodes in tandem into a single node.

Theorem 4 *We can further aggregate the model in Fig*. 3 *into a simpler model as shown in Fig*. 4 *with the equivalent parameters* α, β *and* w *are given as follows*:

$$\tilde{\alpha}_0^1 = \frac{\hat{\alpha}_0^1\hat{\alpha}_1^2}{\hat{\alpha}_1^2 + \hat{\beta}_2^1}, \qquad \tilde{\beta}_2^1 = \frac{\hat{\beta}_3^2\hat{\beta}_2^1}{\hat{\alpha}_1^2 + \hat{\beta}_2^1}, \qquad \tilde{w}_2 = \hat{w}_3,$$

$$\tilde{x}_1^* = \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\alpha}_1^2}\hat{x}_1^* = \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\beta}_2^1}\hat{x}_2^*.$$

Fig. 4 Equivalent 1-node representation



Proof We first consider the mapping from Fig. 3 to Fig. 4. Suppose $(\hat{x}_1^*, \hat{x}_2^*)$ is the optimal solution to Fig. 3. We then obtain $\tilde{x}_1^*$ and show that it is a feasible solution to Fig. 4.

$$\tilde{\alpha}_0^1 \tilde{x}_1^* = \frac{\hat{\alpha}_0^1 \hat{\alpha}_1^2}{\hat{\alpha}_1^2 + \hat{\beta}_2^1} \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\alpha}_1^2} \hat{x}_1^* = \hat{\alpha}_0^1 \hat{x}_1^* \le \lambda_0$$

$$\tilde{x}_1^* = \hat{x}_1^* + \hat{x}_2^* \le R$$

For the mapping from Fig. 4 to Fig. 3, besides the two equations above, we need to check the flow constraint,

$$\hat{\alpha}_1^2 \hat{x}_2^* = \frac{\hat{\alpha}_1^2 \hat{\beta}_2^1}{\hat{\alpha}_1^2 + \hat{\beta}_2^1} \tilde{x}_1^* = \hat{\beta}_2^1 \hat{x}_1^*.$$

Finally, we check the objective values are the same. Then we can map the optimal solution of one problem to a feasible solution to the other problem and maintain the objective values. Therefore, the mapped solution must be optimal for its corresponding problem.

$$\tilde{w}_2 \tilde{\beta}_2^1 \tilde{x}_1^* = \hat{w}_3 \frac{\hat{\beta}_3^2 \hat{\beta}_2^1}{\hat{\alpha}_1^2 + \hat{\beta}_2^1} \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\beta}_2^1} \hat{x}_2^* = \hat{w}_3 \hat{\beta}_3^2 \hat{x}_2^*.$$

□

Besides binary trees, Proposition 3 can also be applied repeatedly to handle general fork trees with arbitrary out-degree (≥ 2). It is straight forward to check the formula that the result of the merging process in Proposition 3 does not depend on the order of the merging process. It is also straight forward to prove a similar theorem as Theorem 2. The proofs are omitted due to space limitations.

The idea of dealing with general tree is to recursively apply Theorem 2 to each two-layer subtree and replace it with a new node. The following algorithm states the whole process.

Algorithm 2 (Backward shrink algorithm for general trees)

1. If there are 2 leaves with a common predecessor, apply Theorem 3 to these 3 nodes (2 leaves and their predecessor) to find the equivalent 2 node structure. Otherwise, Use Theorem 4 to aggregate 2 nodes (a leave and its predecessor) to be a single node structure.
2. Repeat from step 1 until there is only one node left.
3. Set all resource to that node, and mapping resource allocation back according to either Theorem 3 or Theorem 4.

Theorem 5 *Algorithm 2 terminates and yields the optimal solution. It runs in time* $O(|\mathcal{E}|)$.

Proof Since each round of execution of step 1 decreases the number of links by 1, the complexity is $O(|\mathcal{E}|)$.

We now prove optimality using induction. When we have only one node, it is clear that the algorithm generates the optimal solution. Suppose the algorithm generates the optimal solution for all the trees with less than K nodes. Now let's consider a tree with $K+1$ nodes. We first use the algorithm to generate our solution $x^{\#}$. Suppose the optimal solution to the original solution, x^* is strictly better than $x^{\#}$. We will then derive contradictions.

Algorithm 2, solves a series of simple problems like Figs. 2 and 3 to merge the leaf nodes all the way up to the root node. One of the last steps in the merging process results in a fork graph with the root and its children only. Each of its children is an aggregation of its descendents. From $x^{\#}$ and x^*, we can obtain the corresponding solutions for this fork graph by aggregation. Based on the induction hypothesis, if the corresponding solutions are the same for this fork graph, then $x^{\#}$ and x^* must be the same. If the corresponding solutions are different, then our results for the fork graph imply that there cannot be another solution which is strictly better than the solution from Algorithm 2. Therefore, x^* can not be strictly better than $x^{\#}$, which then yields a contradiction. Hence, $x^{\#}$ must be an optimal solution. By induction, Algorithm 2 generates an optimal solution for general trees with any number of nodes. □

6 General aggregation rules

In this section, we discuss general rules that can be applied to aggregate some special subgraphs into equivalent *super nodes*. Such aggregation rules can be applied to arbitrary network topology and help reduce substantially the size of the original systems. Since the optimal resource allocation for the nodes inside a supernode is determined locally, subject to the total available resource to that super node, one can consider all nodes in a subgraph as a cluster and the role of super node can be played by an elected local leader. The leader can then represent the whole cluster and negotiate with other leaders. Such distributed and local management is also more robust especially when the system parameters are not static and subject to changes. Therefore, instead of requiring a centralized and global optimization, it enables a distributed way of resource management.

We first present some general properties of the centralized solutions for the resource allocation problem, and then explore special subgraphs that can be represented by supernodes.

6.1 Series-parallel subgraphs

We say that a graph is a two-terminal labeled graph (G, s, t) with marked head node(terminal) s and tail node(terminal) t, if for any node $k \in G$, we have $s \prec k \prec t$. The series composition of two-terminal labeled graphs (G, s, t) and (H, x, y) with $t = x$ is $(G \cup H, s, y)$. The parallel composition of (G, s, t) and (H, x, y) with $s = x$ and $t = y$ is $(G \cup H, s, t)$. A graph is a *series-parallel* if it can be created from single two-terminal labeled edges by series and/or parallel compositions. Detailed definitions and characterizations on the subclass of directed *series-parallel graphs*, have been given in Brandstädt et al. (1999). The left plot in Fig. 5 illustrates one such example where node 1 is the head and node N is the tail. For the ease of presentation, in this section, we shall always label the head node as node 1 and the tail node as node N.

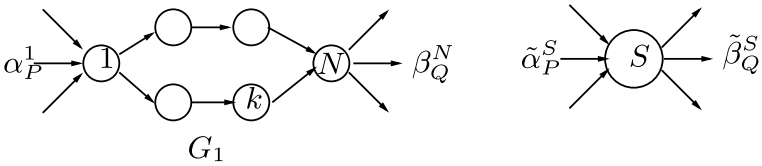


Fig. 5 Series-parallel subgraphs & equivalent supernode

Consider a graph $G = (\mathcal{N}, \mathcal{E})$ that has a series-parallel subgraph $G_1 = (\mathcal{N}_1, \mathcal{E}_1)$. Denote $G_2 = (\mathcal{N}_2, \mathcal{E}_2) := G \setminus G_1$. We claim that subgraph G_1 can be aggregated into an equivalent super node S using the following procedure.

Aggregation Procedure 1 (Series-parallel graphs)

- Apply step 1 and 2 of Algorithm 1 on subgraph G_1. Denote the resulting allocation $\hat{\mathbf{x}}_1 = (\hat{x}_k, k \in G_1)$. Then, under allocation $\hat{\mathbf{x}}$, the required input rates at the node 1 is $\hat{\alpha}_P^1 = \hat{x}_1 \alpha_P^1$, from its parent nodes $P \in \mathcal{I}^1$; in addition, since $\hat{x}_N = 1$, the tail node N produces output at rate $\hat{\beta}_Q^N = \hat{x}_N \beta_Q^N = \beta_Q^N$ to its child nodes $Q \in \mathcal{O}^N$.
- Define a *supernode* S (see the right plot in Fig. 5). Given a unit of computing resource, node S requires inputs at rates $\tilde{\alpha}_P^S := \frac{\hat{\alpha}_P^1}{\hat{x}_S}$, for $P \in \mathcal{I}^1$, and produces output at rates $\tilde{\beta}_Q^S := \frac{\hat{\beta}_Q^N}{\hat{x}_S}$, for $Q \in \mathcal{O}^N$, where $\hat{x}_S := \sum_{k \in G_1} \hat{x}_k$.
- Replace subgraph G_1 (in the original graph G) by supernode S, we obtain a new graph $\tilde{G} = (\tilde{\mathcal{N}}, \tilde{\mathcal{E}})$, where $\tilde{\mathcal{N}} = \{S\} \cup N_2$, and $\tilde{\beta}_S^P = \beta_1^P$ for all $P \in \mathcal{I}^1$.

Theorem 6 *The original* (LP) *problem for graph G is equivalent to the* (LP) *for the new graph $\tilde{G}$ defined in Aggregation Procedure* 1. *If $(\tilde{x}_S, \tilde{\mathbf{x}}_2)$ is the optimal solution of the* (LP) *on the new graph $\tilde{G}$, then $\mathbf{x} := (\tilde{x}_S \frac{\hat{\mathbf{x}}_1}{\hat{x}_S}, \tilde{\mathbf{x}}_2)$ is optimal for the original* (LP) *on graph G.*

Proof See Appendix A for detailed proof. □

The following theorem is immediate.

Theorem 7 *Any series-parallel subgraph (G_1, s, t) in G can therefore be aggregated to an equivalent super node following Aggregation Procedure* 1.

6.2 Single link partition

Suppose link $l = (a, b)$ has the property that by removing l, the original graph G can be partitioned into two disjoint subgraphs G_1 and G_2, with $a \in G_1$ and $b \in G_2$. We call (G_1, G_2) a single link (a, b) partition for G. Fig. 6 illustrates such an example. In this section, we derive some basic properties of the optimal solution on such links and present a method that can aggregate subgraph G_2 into an equivalent super node S.

The next theorem shows that if node b is active in the optimal solution, it then must fully utilize all the output flow produced by node a.

Theorem 8 *Suppose graph G has a single link (a, b) partition (G_1, G_2) with $a, b \in \mathcal{P}$. There must be an optimal solution $\mathbf{x}^*$ for the* (LP) *on graph G such that either $x_b^* = 0$ or*

$$\alpha_a^b x_b^* = \beta_b^a x_a^*. \tag{4}$$

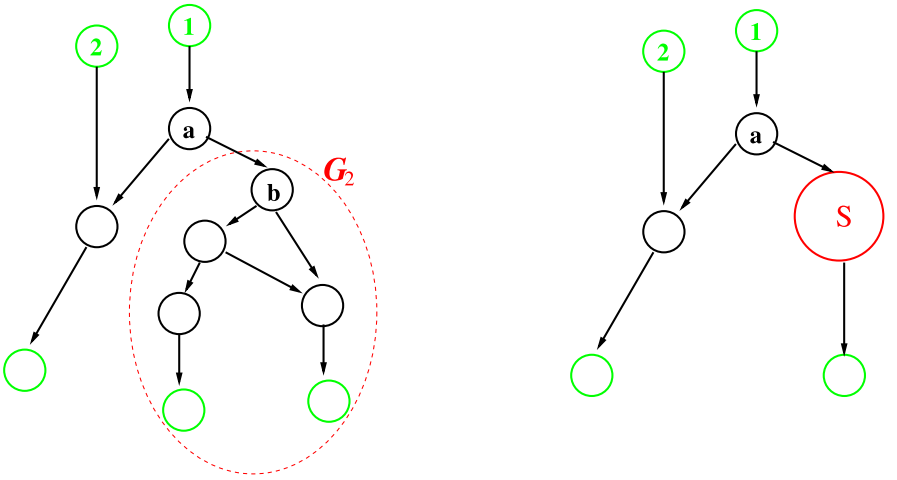


Fig. 6 Single link partition & aggregation

Proof See Appendix B for detailed proof. □

Consider the special case when the graph G is a tree. In this case, any link in $\mathcal{E}$ can produce a single link partition for G. The following corollary is then immediate.

Corollary 2 *For the* (LP) *where the underlying graph G is a tree, there must be an optimal solution x^* such that for link (i, j) is in $\mathcal{E}$ with $i, j \in \mathcal{P}$, either $x_j^* = 0$, or $\alpha_i^j x_j = \beta_j^i x_i$. That is, either node j is idle, or else it must fully utilize all the output flow from its parent node i.*

We then have a procedure that aggregates the subgraph G_2 into an equivalent super node S.

Aggregation Procedure 2 (Single-link partitioned graphs)

1. First formulate (LP) for G_2 assuming a is the source node with maximum input rate available to node b is 1, and there is no capacity constraint. Solve for the optimal solution $\hat{\mathbf{x}}_2 = (\hat{x}_j, j \in G_2)$. Suppose it produces benefit $V_2(\hat{\mathbf{x}}_2)$. Denote $\hat{x}_S := \sum_{j \in G_2} \hat{x}_j$.
2. Represent subgraph G_2 by a super-node S. With a unit of computing resource, node S requires input from node a at rate $\tilde{\alpha}_a^S = \frac{1}{\hat{x}_S}$, and produces a single output stream at rate $\tilde{\beta}_O^S = \frac{V_2(\hat{\mathbf{x}}_2)}{\hat{x}_S}$ with weight $w_O = 1$. Denote the new graph as $\tilde{G} = G_1 \cup \{S\}$, where $\tilde{\beta}_S^a = \beta_b^a$.
3. Formulate LP for new graph $\tilde{G}$. Solve for the optimal solution $(\tilde{\mathbf{x}}_1, \tilde{x}_S)$ where $\tilde{\mathbf{x}}_1 = (\tilde{x}_j, j \in G_1)$.
4. Obtain final solution $\tilde{\mathbf{x}} = (\tilde{\mathbf{x}}_1, \tilde{x}_S \frac{\hat{\mathbf{x}}_2}{\hat{x}_S})$.

Theorem 9 *The original* (LP) *problem for graph G is equivalent to the* (LP) *for the new graph $\tilde{G} = \{S\} \cup G_2$. If $(\tilde{\mathbf{x}}_1, \tilde{x}_S)$ is the optimal solution of the* (LP) *on the new graph $\tilde{G}$, then $\mathbf{x} := (\tilde{\mathbf{x}}_1, \tilde{x}_S \frac{\hat{\mathbf{x}}_2}{\hat{x}_S})$ is optimal for the original* (LP) *on graph G.*

Proof See Appendix C for detailed proof. □

Notice that when the network is a tree, any link can produce a single link partition for G. One can therefore shrink any subtree into an equivalent super node. The backward shrink algorithm presented in Sect. 5 is just doing this backward recursively. The following corollary is hence immediate based on Theorem 9.

Corollary 3 *The backward shrink algorithm for networks of a tree structure must produce an optimal solution to the original problem.*

The above aggregation rules can be applied to general network topologies so as to enable a distributed ways of resource management. In the next section, we present two specific distributed solutions for the general problem.

7 Distributed solutions

In this section, we present distributed solutions for the general problem. Simulation experiments demonstrate that they perform well even for general network topologies that do not have a tree structure, and for systems with non-stationary input traffic and with dynamically changing parameters.

7.1 Distributed algorithms

We develop two heuristics to solve the general problem. These heuristics are based on the optimal solutions for the tree case and for the single-output case. Experimental results are provided to illustrate their effectiveness. As we will see in the next section, these heuristics can be implemented easily in a distributed way.

The first heuristic is based on the optimal solution for trees. As assumed earlier, all the nodes have been labeled from 1 to N such that all the edges (i, j) satisfy $i < j$. This algorithm will start from the bottom of the graph and move up to the top. At each step, the algorithm examines each node, generate aggregated information based on information from its children, and pass this information up to its parents.

Heuristic A

- initialize graph G to be the whole graph;
- for $node = N$ to 1 (compute bottom up for the aggregated solution)
  - if *node* is a leaf in G then pass its parameters α, β, w to its parents;
  - else (all the children of *node* must be leaves in G;)
    - if *node* has more than one child, apply Theorem 3 repeatedly to remove one leaf at a time from G;
    - apply Theorem 4 to obtain the updated parameters α, β, w for *node*;
    - pass the updated parameters to all its parents;
      (*node* has no children left in G;)
- G has one node left, with aggregated parameters;
- solve this single node problem;
- for $node = 1$ to N (compute a solution for original problem from top down)
  - apply Theorem 2 and 4 to compute solution for *node* and the flow amount to all its children;

The key idea of Heuristic A is to treat each node with more than one children as a two level tree and use Theorem 3 to shrink the tree into an aggregated node. Iterate this procedure bottom up until the whole graph is aggregated to a single node.

If the original graph is a tree, it can be shown that the above algorithm obtains the optimal solution. For the general graph case, we will present experimental results to demonstrate the quality of this distributed algorithm.

Another heuristic for the general problem with multiple output streams is developed based on the single output algorithm combined with the general gradient decent algorithm. Assume there are multiple output streams, $O_1, \ldots, O_k$. We define a function $f(u_1, \ldots, u_k)$ to be the best objective value if the solutions are generating flows for the output streams according to the relative proportion given by $(u_1, \ldots, u_k)$. Finding $f(u_1, \ldots, u_k)$ is the same as solving a modified problem with a new final sink node O_{k+1}, and making all the original output flows to flow into this final sink node. The β parameters for all the flows from $O_1, \ldots, O_k$ to O_{k+1} are all set to be 1. The α proportions at O_{k+1} are given by $(u_1, \ldots, u_k)$ for flows from $O_1, \ldots, O_k$. The β parameter at O_{k+1} is $w_1u_1 + \ldots + w_ku_k$. The weight factor w at O_{k+1} is 1. The equivalence of these two problems can be easily checked. Since we can apply the backtrack algorithm in the earlier sections to find the optimal solution for the single output problem, we can find the value of $f(u_1, \ldots, u_k)$ for any given $(u_1, \ldots, u_k)$. We now apply the gradient decent algorithm to find the maximum value for function $f(u_1, \ldots, u_k)$.

Heuristic B

(0) Node O_{k+1} initializes $(u_1, \ldots, u_k)$ to be $(w_1, \ldots, w_k)$;
(1) Run Distributed Algorithm 1 across the system with output target $(u_1, \ldots, u_k)$;
(2) Node O_{k+1} estimates the gradient for $f(u_1, \ldots, u_k)$;
(3) Node O_{k+1} moves point $(u_1, \ldots, u_k)$ along the gradient direction;
(4) Repeat from step (1) until relative difference between consecutive solutions is smaller than a given threshold.

Note that in Heuristic B, the gradient method can be replaced by other search techniques such as simulated annealing, Tabu search, genetic algorithms, smart hill-climbing (Xi et al. 2004), etc.

Heuristic A has the advantage that it can quickly generate high quality solutions for simple graph topologies. However, when the graph is complex, the quality may degrade. Heuristic B is expected to be able to handle more effectively complex graph structures.

7.2 Experimental results

We present below experimental results to compare the performance of these two heuristics and the optimal solution. The setting of the experiment is as follows. First, directed acyclic graphs with N nodes are generated randomly using the following 4 steps:

(1) Randomly generate N points (x_i, y_i) in the unit square $[0, 1] \times [0, 1]$;
(2) For $i = 1, \ldots, N$, generate its successor set $S_i := \{j : x_j \geq x_i, y_j \geq y_i\}$;
(3) For $i = 1, \ldots, N$, generate its immediate successor set $s_i := S_i - \bigcup_{k \in S_i} S_k$;
(4) For $i = 1, \ldots, N$, create a link from i to j if $j \in s_i$.

This algorithm is inspired by a scheme to generate random partial orders among N elements. Once the graph is generated, the parameters α, β, w are then generated from independent uniform random samples.

Table 1 Results of Heuristic A for networks of random topology

# of nodes	20	50	100
Avg # of edges	74.2	507.4	2063.7
Avg # of source nodes	4.6	5.8	7.0
Avg # of sink nodes	3.1	3.7	4.9
% optimality (avg)	74.4	57.6	54.3
% optimality (std)	26.6	33.2	34.3
% cases >90% optimal	42.1	29.5	25.1

Table 2 Results of Heuristic B for networks of random topology

# of nodes	20	50	100
Avg # of edges	79.1	520.1	1912.7
Avg # of source nodes	4.4	4.9	7.6
Avg # of sink nodes	3.4	3.6	5.0
% optimality (avg)	82.4	68.9	59.0
% optimality (std)	25.4	36.4	39.2
% cases >90% optimal	61	41.1	32.2
Avg # of iterations	5.2	10.1	10.0

We randomly generate graphs with 20, 50, and 100 nodes. For each fixed number of nodes, we generate 1000 instances of the problem with random topology and random parameter values. We apply the two heuristics to obtain the corresponding objective values. We also obtain the optimal solution through a static linear program formulation. We have collected the characteristics of the random graphs, as well as the quality of the two heuristics. Because the problem is a maximization problem, the quality of the heuristics is reflected by the achieved percentage of the optimal solution. The results from Heuristic A are presented in Table 1. We can see that Heuristic A generates reasonably good solutions for small size graphs. However, the quality of the solutions degrades as the size of the graph grows. This behavior is consistent with our earlier intuitions.

Table 2 presents the results for Heuristic B. We used 10% relative difference as the stopping criterion for the gradient algorithm. We observe that Heuristic B is consistently better than Heuristic A. It is also important that the average number of iterations is small. This means Heuristic B does not require too much additional time to compute compared with Heuristic A. It is very promising to find out that Heuristic B consistently generates quality solutions, and more importantly, its effectiveness can be improved through the use of more sophisticated search methods. Keeping in mind that we are interested in the distributed nature and the efficiency of the algorithm. Heuristic B seems to be a preferable solution.

7.3 Dynamically changing environment & non-stationary systems

As we mentioned at the beginning of the paper, the stream processing environment (e.g., flow properties and system parameters) can be dynamically changing. Furthermore, the stream processing systems can have non-stationary behaviors: the input streams can be non-stationary in the traffic patterns and/or in contents; the stream consumption and production can be non-stationary due to the non-stationary contents. Therefore, the resource allocation solutions should adapt to the changes.

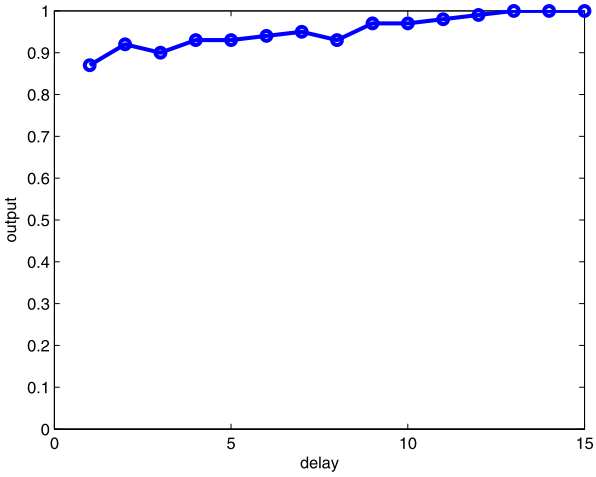


Fig. 7 Achieved performance ratio (compared to the optimal) over time: for random tree networks with perturbation at time 0

The distributed implementations of the heuristics described above can easily handle such changes by local adaptation. In fact, each node will constantly (with, say fixed, sampling frequency) measure the input rates, the consumption rates and the production rates. If significant deviation occurs, which can be detected with some change-point detection techniques, the node will send the updated rate information to parents. These parents will in turn forward the updated, aggregated parameters to their parents up to the top node which will readjust the resource allocation decisions and propagate the resource allocation readjustments downstream. To avoid abrupt changes in the resource allocation decisions, we add a smoothing step in both distributed algorithms such that the actual resource allocated for each node is the moving average of its previous allocation and the current new assignment with a preselected smoothing factor γ.

It is easily observed from Heuristic A that when a changing event occurs, it takes twice the depth of the graph for the system to adjust to the new optimal solution. Figure 7 illustrates such a scenario for a random tree network. Specifically, a random tree structure with 355 nodes is generated where the depth equals to 7. At time 0, system is running at the optimal solution, but then a random subset of nodes are selected and the corresponding parameters are perturbed by $\pm 20\%$, hence the system needs to find the new optimal solution. Figure 7 shows that as time (here we assume a unit time is the propagation delay for one hop, thus t time units means t hops away) proceeds, the performance of the perturbed system gradually converges and actually reaches the new optimal at time $t = 14$ which is twice the depth of the tree. Here the smoothing factor $\gamma = 0.5$. Table 3 shows the average time it takes for the perturbed system to reach 95% of the new optimal over 1000 random instances of the problem for random tree networks with 50, 100, 500 and 1000 nodes respectively (depth fixed to be 7). We see that although it takes 14 time units for the distributed algorithm to converge to the new optimal, the algorithm is able to quickly adapt to changed environment and reaches 95% of the optimal in much shorter time.

To illustrate the performance of the distributed algorithm in non-stationary systems, we use the same examples as above, but now the system is perturbed every 5 time units, each perturbation involves a random subset of the nodes with the corresponding parameters varying by $\pm 20\%$. In Fig. 8, the solid line gives the new optimal return associated with each perturbation, which clearly shows a system that is non-stationary. The performance under our distributed algorithm is given by the dashed line, which is quite close to the new optimal.

Table 3 Average time to reach 95% of optimal: for random tree networks with perturbation at time 0

# of nodes	50	100	500	1000
Avg time reaching 95% of optimal (in time units)	4.1	4.8	5.4	7.1

Fig. 8 Achieved output value over time: for random tree networks with perturbation every 5 time units

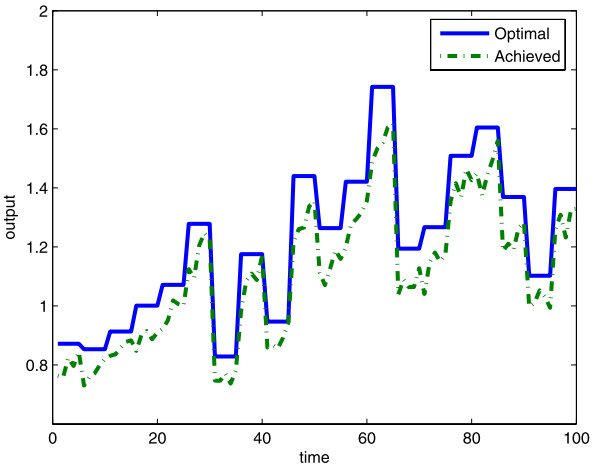


Table 4 Average achieved performance: for random tree networks with perturbation every 5 time units

# of nodes	50	100	500	1000
Avg achieved performance (in percentage) compared to optimal	96.2	94.2	91.9	90.4

Table 4 shows the average achieved performance (in percentage) compared to the optimal over 1000 random instances of the non-stationary system for random tree networks with 50, 100, 500 and 1000 nodes respectively, where perturbation happens continuously every 5 time units. We see that our distributed algorithm quickly adapts to the continuous changes and is able to deliver high quality solutions.

8 Concluding remarks

This paper solves the CPU resource allocation problem in stream processing systems with the objective of maximizing the total return of multiple output streams. We explore structural properties of the optimal solution for the problem under different network topologies, and develop efficient, yet simple to implement algorithms to solve them. Detailed performance analysis on optimality and complexity of those algorithms are also provided.

We further present two distributed solutions to the general problem and give the corresponding measurement-based distributed implementation. Our experimental results show that the algorithms are highly robust. The algorithms quickly adapt to real-time fluctuations

in the consumption and production rates and changes in resource consumption requirements, and can achieve high quality solutions even in non-stationary systems.

Appendix A: Proof for Theorem 6

It suffices to show that from any optimal solution $\tilde{\mathbf{x}}$ for the (LP) problem on graph $\tilde{G}$, we can find a feasible solution $\mathbf{x}$ for the LP problem on graph G that uses the same amount of capacity and has the same objective value. Similarly, from any optimal solution for (LP) on graph G, we can find a feasible solution for the (LP) problem on the graph $\tilde{G}$, that uses the same amount of capacity and has the same objective value.

First we show $\mathbf{x} = (\tilde{x}_S \frac{\hat{\mathbf{x}}_1}{\hat{x}_S}, \tilde{\mathbf{x}}_2)$ is feasible to the original (LP) on graph G. For all $P \in \mathcal{I}^1$, the internal flow constraints on links $(P, 1)$ in graph G are satisfied because $\tilde{x}_S \frac{\hat{x}_1}{\hat{x}_S} \cdot \alpha_P^1 = \tilde{x}_S \tilde{\alpha}_P^S \le \tilde{x}_P \tilde{\beta}_S^P = \tilde{x}_P \beta_1^P$, where the inequality holds because $(\tilde{x}_S, \tilde{\mathbf{x}}_2)$ satisfies the internal flow constraint on link (P, S) on graph $\tilde{G}$. Similarly one can verify the internal flow constraints on links (N, Q) for all $Q \in \mathcal{O}^N$. Therefore $\mathbf{x}$ must be feasible to the original (LP) on graph G.

Suppose $\mathbf{x}^*$ is optimal to the (LP) for graph G. Denote $\mathbf{x}_1^* = (x_j^*, j \in G_1)$, $\mathbf{x}_2^* = (x_j^*, j \in G_2)$. The amount of output flow at node N under allocation $\mathbf{x}^*$ is then $B_Q^N = x_N^* \beta_Q^N$, for all $Q \in \mathcal{O}^N$. Let $x_S^* = \sum_{j \in G_1} x_j^*$. We claim that $(x_S^*, \mathbf{x}_2^*)$ is also feasible to the (LP) on new graph $\tilde{G}$.

Denote $\gamma = x_N^*$. Based on Lemma 4, $\gamma \hat{\mathbf{x}}_1$ is the solution with minimum total resource requirement $\gamma \hat{x}_S$ and minimum input rate requirements $\gamma \hat{\alpha}_P^1$, $P \in \mathcal{I}^1$ in order to produce output at rates $\gamma \hat{\beta}_Q^N = B_Q^N$ for all $Q \in \mathcal{O}^N$. Since $\mathbf{x}_1^*$ also produces B_Q^N for all $Q \in \mathcal{O}^N$, we must have:

$$x_1^* \alpha_P^1 \ge \gamma \hat{\alpha}_P^1 = \gamma \hat{x}_1 \alpha_P^1, \quad \forall P \in \mathcal{I}^1, \tag{5}$$

$$x_S^* \ge \gamma \hat{x}_S = \gamma \sum_{k \in G_1} \hat{x}_k. \tag{6}$$

We claim that both (5) and (6) must be tight. If (6) is not tight, one can replace $\mathbf{x}_1^*$ by $\gamma \hat{\mathbf{x}}_1$, distribute the remaining capacity proportionally, and obtain a solution $\mathbf{x} = (1 + \epsilon)(\gamma \hat{\mathbf{x}}_1, \mathbf{x}_2^*)$ (for ϵ sufficiently small) that performs better than $\mathbf{x}^*$, which contradicts that $\mathbf{x}^*$ is optimal. If (6) is tight, but (5) is not, then $x_1^* > \gamma \hat{x}_1$, and we must have some k ($k \ne 1$) such that $x_k^* < \gamma \hat{x}_k$, which contradicts that $\gamma \hat{x}_k$ is the minimum required resource at node k to produce $\{\gamma \beta_Q^N,\ Q \in \mathcal{O}^N\}$ based on Lemma 4.

By setting both (5) and (6) to equalities, we then have: $x_S^* \tilde{\alpha}_P^S = \gamma \hat{x}_S \frac{\hat{\alpha}_P^1}{\hat{x}_S} = x_1^* \alpha_P^1 \le x_P^* \beta_1^P = x_P^* \tilde{\beta}_S^P$, $\forall P \in \mathcal{I}^1$, where the inequality is because $\mathbf{x}^*$ must satisfy the internal flow constraint on link $(P, 1)$. Therefore, $(x_S^*, \mathbf{x}_2^*)$ also satisfies the internal flow constraint on link (P, S) and it must be feasible to the (LP) on new graph $\tilde{G} = \{S\} \cup G_2$.

Therefore the two approaches must be equivalent. □

Appendix B: Proof for Theorem 8

Consider an optimal solution $\mathbf{x}^*$ for the (LP) on graph G. Denote $\mathbf{x}_i^* = (x_j^*, j \in G_i)$, $T_i^* = \sum_{j \in G_i} x_j^*$, and V_i^* the partial objective sum corresponding to $\mathbf{x}_i^*$, $i = 1, 2$. The total objective value is $V^* = V_1^* + V_2^*$.

Suppose $x_b^* > 0$, and (4) does not hold, then $\gamma := \frac{\beta_b^a x_a^*}{\alpha_a^b x_b^*} > 1$. We must have:

$$\frac{V_1^*}{T_1^*} < \frac{V_2^*}{T_2^*}. \tag{7}$$

Otherwise, the return per unit of resource is higher on G_1 than that on G_2, we can re-allocate the total resource T_2^* to G_1 (proportional to $\mathbf{x}_1^*$) and obtain a better solution with $x_b = 0$.

Given (7), we can use a new allocation $\mathbf{x} = (\mathbf{x}_1^*, \gamma \mathbf{x}_2^*)\frac{T_1^*+T_2^*}{T_1^*+\gamma T_2^*}$, which not only satisfies (4) but also has a better return since $V^* = V_1^* + V_2^* < (V_1^* + \gamma V_2^*)\frac{T_1^*+T_2^*}{T_1^*+\gamma T_2^*}$. This contradicts the optimality of $\mathbf{x}^*$. Therefore we must have (4) or else $x_b^* = 0$. □

Appendix C: Proof for Theorem 9

It suffices to show that from any optimal solution for (LP) on the original graph G, we can find a feasible solution for the (LP) problem on the new graph $\tilde{G}$, that uses the same amount of capacity and has the same objective value. Similarly, from any optimal solution for the (LP) problem on the new graph $\tilde{G}$, we can find a feasible solution for the (LP) problem on the original graph G that uses the same amount of capacity and has the same objective value.

Let $\mathbf{x}^*$ be an optimal solution to the original (LP) problem for the original graph G, and the corresponding objective value $V(\mathbf{x}^*)$. Denote $\mathbf{x}_i^* = (x_j^*, j \in G_i)$, and $V_i(\mathbf{x}_i^*)$ the partial objective sum corresponding to $\mathbf{x}_i^*$, $i = 1, 2$. Let $x_S^* = \sum_{j\in G_2} x_j^*$. We show that $\tilde{\mathbf{x}} := (\mathbf{x}_1^*, x_S^*)$ is also a feasible solution to the (LP) on new graph $\tilde{G} = (G_1, S)$.

Clearly all internal flow constraints on subgraph G_1 are satisfied by $\tilde{\mathbf{x}}$. For $\tilde{\mathbf{x}}$ to be feasible on $\tilde{G}$, it suffices to show to check the flow constraint on the link from node a to super node S, i.e., we need:

$$\tilde{\beta}_S^a x_a^* \geq \tilde{\alpha}_a^S x_S^*. \tag{8}$$

Denote $\lambda^* := \alpha_a^b x_b^*$. Then $\mathbf{x}_2^*/\lambda^*$ is feasible to the (LP) on subgraph G_2, which requires a unit input from node a, and produces benefit $V(\mathbf{x}_2^*)/\lambda^*$. Since $\hat{\mathbf{x}}_2$ is the best solution to the (LP) on subgraph G_2 given a unit input from node a, we must have $V_2(\mathbf{x}_2^*)/\lambda^* \leq V_2(\hat{\mathbf{x}}_2)$.

In addition, we claim that $x_S^*/\lambda^* \leq \hat{x}_S = \sum_{j\in G_2} \hat{x}_j$. Otherwise, if $x_S^*/\lambda^* > \hat{x}_S$, let $\delta = x_S^* - \lambda^* \hat{x}_S (> 0)$, then $\mathbf{x} = \frac{1}{1-\delta}(\mathbf{x}_1^*, \lambda^* \hat{\mathbf{x}}_2)$ is also feasible to the original problem (LP) on graph G, and has objective value $V(\mathbf{x}) = \frac{1}{1-\delta}(V_1(\mathbf{x}_1^*) + \lambda^* V_2(\hat{\mathbf{x}}_2)) \geq \frac{1}{1-\delta}(V_1(\mathbf{x}_1^*) + V_2(\mathbf{x}_2^*)) > V(\mathbf{x}^*)$, which contradicts that $\mathbf{x}^*$ is optimal.

Hence (8) must hold because $\tilde{\beta}_S^a x_a^* = \beta_b^a x_a^* \geq \alpha_a^b x_b^* = \lambda^* \geq \frac{x_S^*}{\hat{x}_S} = \tilde{\alpha}_a^S x_S^*$. Therefore, $(\mathbf{x}_1^*, x_S^*)$ is a feasible solution to the (LP) on new graph $\tilde{G}$, and we have

$$V(\mathbf{x}^*) = \tilde{V}((\mathbf{x}_1^*, x_S^*)) \leq \tilde{V}\left(\left(\tilde{\mathbf{x}}_1, \tilde{x}_S \frac{\hat{\mathbf{x}}_2}{\hat{x}_S}\right)\right). \tag{9}$$

It is easily checked that $\tilde{\mathbf{x}} = (\tilde{\mathbf{x}}_1, \tilde{x}_S \frac{\hat{\mathbf{x}}_2}{\hat{x}_S})$ is feasible to the original (LP) problem on graph G. This is because $\alpha_a^b \hat{x}_b = 1$ and $\beta_S^a \tilde{x}_a \geq \tilde{\alpha}_a^S \tilde{x}_S / \hat{x}_S = \tilde{x}_S / \hat{x}_S = \alpha_a^b (\tilde{x}_S \frac{\hat{x}_b}{\hat{x}_S})$. Thus

$$\tilde{V}\left(\left(\tilde{\mathbf{x}}_1, \tilde{x}_S \frac{\hat{\mathbf{x}}_2}{\hat{x}_S}\right)\right) = V(\mathbf{x}) \leq V(\mathbf{x}^*). \tag{10}$$

Combine (9) and (10), we then have the results as claimed. □

References

Baccelli, F., & Liu, Z. (1989). On the stability condition of a precedence-based queueing discipline. *Advances in Applied Probability*, *21*, 883–887.

Baccelli, F., & Liu, Z. (1990). On the execution of parallel programs on multiprocessor systems—a queuing theory approach. *Journal of the Association for Computing Machinery*, *37*(2), 373–417.

Baldwin, C., Clark, K. B., Magretta, J., Dyer, J. H., Fisher, M., & Fites, D. V. (2000). *Harvard business review on managing the value chain*. Boston: Harvard Business School Press.

Brandstädt, A., Bang Le, V., & Spinrad, J. P. (1999). *Graph classes: a survey*. Philadelphia: SIAM.

Mas-Collel, A., Whinston, M., & Green, J. (1995). *Microeconomic theory*. London: Oxford University Press.

Simchi-Levi, D., Kaminsky, P., & Simchi-Levi, E. (2002). *Designing and managing the supply chain* (2nd ed.). New York: McGraw-Hill.

Viglas, S., & Naughton, J. (2002). Rate-based query optimization for streaming information sources. In *ACM SIGMOD*.

Xi, B., Liu, Z., Raghavachari, M., Xia, C., & Zhang, L. (2004). A smart hill-climbing algorithm for application server configuration. In *Proc. of the 13th international conference on world wide web* (pp. 287–296).