Distributed Resource Allocation for Stream Data Processing

Ao Tang[1], Zhen Liu[2], Cathy Xia[2], and Li Zhang[2]

[1] California Institute of Technology, Department of Electrical Engineering, Pasadena, CA 91125
[2] IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532

Abstract. Data streaming applications are becoming more and more common due to the rapid development in the 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. It is critical to optimize the ongoing 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. Data streams flow through such a system after processing at multiple processing units. Based on this framework, we develop distributed algorithms for finding the best resource allocation schemes in such data stream processing networks. Performance analysis on the optimality and complexity of these algorithms are also provided.

Keywords: Stream Processing, Distributed Algorithm, Resource Allocation.

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 applications 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 information 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

M. Gerndt and D. Kranzlmüller (Eds.): HPCC 2006, LNCS 4208, pp. 91–100, 2006.

be processed in real-time. Sensor networks is another area where many stream data processing applications arise. The nature of these continuous processing applications to process a large volume of data has lead to many new 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 resource. 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 some of the fundamental resource allocation problems raised above. We formulate a generic stream data processing model with data streams passing through multiple processing units to generate the result streams. Sub-optimal allocation of the resources may lead to the under-utilization of certain processing units and over-utilization of some others. Our goal is to obtain a distributed 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 and supply chain management [1,2,4,10,8]. It is an important feature in many streaming processing applications. For example, the network usage information from multiple routers need 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 query like applications [12].

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. We then investigate the structural properties of the optimal solutions for special cases (a single output stream case and the tree case) in Sections 3. In Section 4, 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 Section 5.

2 Model

In a stream data processing system, incoming data flow 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, 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, where 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 Figure 1.

Denote $\mathcal{I}, \mathcal{P}, \mathcal{O}$ respectively the set of source, processing, and sink nodes in the graph, as illustrated in Figure 1. Let $\mathcal{E}$ denote the set of all the directed edges. Each node in $\mathcal{I}$ is a source node. Each node in $\mathcal{O}$ is a sink node. For convenience, we will refer to the underlying graph, $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \mathcal{I} \cup \mathcal{P} \cup \mathcal{O}$, and graph G is assumed to be 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 $k \in \mathcal{O}$.

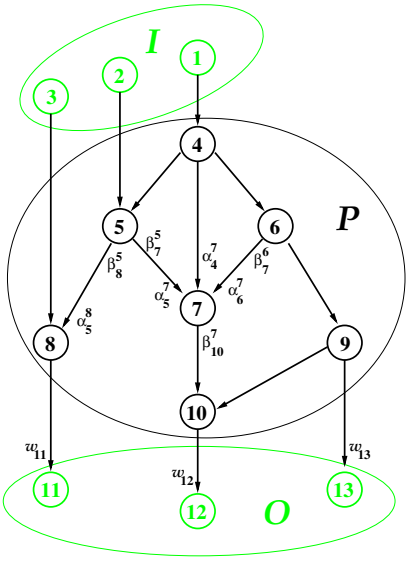

Fig. 1. A graph representation of the problem

We now describe the quantitative relationship between the input, output and resource consumption. In our model, each processing unit processes data flows from its upstream nodes *simultaneously* at a given proportion and 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 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 Section 4.

Assume we have a total of R units of CPU resource available. Our goal is to find optimal or approximate solutions of 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 look for distributed solutions capable of adapting to local changes in the consumption and production rates.

3 The Single Output Case and Trees

We first 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 in time $O(|\mathcal{E}|)$. Please refer to [11] for details of the proof using a backward tracing argument.

Algorithm 1. Graphs with Single Output
1. Initialize set $A = \{N\}$, and let $x_N = 1$.
2. Let $B := \bigcup_{i \in A} \mathcal{I}^i$ be the set of all predecessors of nodes in A.
   - If $B \subset \mathcal{I}$, go to step 3;
   - Else, let $x_i = \max_{\{j \in A:(i,j) \in \mathcal{E}\}} \frac{\alpha_i^j x_j}{\beta_j^i}, \forall i \in B$; set $A = B$; go back to step 2.
3. Let $\mathbf{x} = (x_j, j \in \mathcal{P})$ be the allocation produced by steps 1 & 2. Denote $\delta_{\max} := \max\{\delta > 0 : \delta \alpha_i^j x_j \leq \lambda_i, i \in \mathcal{I}, (i,j) \in \mathcal{E}\}$. Then the final allocation $\mathbf{x}^*$ is given by $x_i^* = \min(\delta_{\max}, \frac{R}{\sum_i x_i}) \cdot x_i, \quad i \in \mathcal{P}$, and the total return is $V^* = \min(\delta_{\max}, \frac{R}{\sum_i x_i}) \beta_O^N$.

We now generalize the previous algorithm to address the cases with multiple output nodes, i.e., $|\mathcal{O}| > 1$. In this setting, there is a decision between generating output for one stream versus generating output for another stream, or both. This kind of trade-off is not easy 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.

Example 2: A Binary Tree
We now define a linear program as follows,

$$\begin{aligned} \max\ & w_4\beta_4^2 x_2 + w_5\beta_5^3 x_3 \\ \text{s.t.}\ & x_1 + x_2 + x_3 \le R, \quad \alpha_0^1 x_1 \le \lambda_0, \\ & \alpha_1^2 x_2, \le \beta_2^1 x_1, \qquad \alpha_1^3 x_3 \le \beta_3^1 x_1 \quad (1) \\ & x1 \ge 0, \quad x2 \ge 0, \quad x_3 \ge 0. \end{aligned}$$

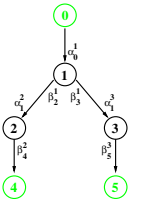

Fig. 2. A 3-node binary tree

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 1. *The solution to the above problem is*
i) If $\frac{w_4\beta_4^2\beta_2^1}{\alpha_1^2+\beta_2^1} > w_5\beta_5^3$, *then,* $x_2^\# = \frac{\beta_2^1}{\alpha_1^2+\beta_2^1}r$, $x_3^\# = 0$, $x_1^\# = \frac{\alpha_1^2}{\alpha_1^2+\beta_2^1}r$, *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,* $x_2^\# = 0$, $x_3^\# = \frac{\beta_3^1}{\alpha_1^3+\beta_3^1}r$, $x_1^\# = \frac{\alpha_1^3}{\alpha_1^3+\beta_3^1}r$, *where* $r = \min(R, \frac{\alpha_1^3+\beta_3^1}{\alpha_0^1\alpha_1^3}\lambda_0)$.
iii) Else, $x_2^\# = \frac{\alpha_1^3\beta_2^1}{\alpha_1^2\alpha_1^3+\alpha_1^3\beta_2^1+\alpha_1^2\beta_3^1}r$, $x_3^\# = \frac{\alpha_1^2\beta_3^1}{\alpha_1^2\alpha_1^3+\alpha_1^3\beta_2^1+\alpha_1^2\beta_3^1}r$, $x_1^\# = \frac{\alpha_1^2\alpha_1^3}{\alpha_1^2\alpha_1^3+\alpha_1^3\beta_2^1+\alpha_1^2\beta_3^1}r$, *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. This result can be proved case by case with linear algebra using contradiction techniques. Please refer to [11] for details. □

Theorem 2. *The problem in Figure 2 is equivalent to the simpler model in Figure 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* $\hat\alpha_0^1 = \alpha_0^1, \hat\beta_2^1 = \beta_2^1$, $\hat\alpha_1^2 = \alpha_1^2, \hat\beta_3^2 = \beta_4^2$, $\hat w_3 = w_4$.
ii) If $\frac{w_5\beta_5^3\beta_3^1}{\alpha_1^3+\beta_3^1} > w_4\beta_4^2$, *then,* $\hat\alpha_0^1 = \alpha_0^1, \hat\beta_2^1 = \beta_3^1$, $\hat\alpha_1^2 = \alpha_1^3, \hat\beta_4^2 = \beta_5^3$, $\hat w_3 = w_5$.
iii) Else, $\hat\alpha_0^1 = \alpha_0^1, \hat\beta_2^1 = \beta_2^1 + \beta_3^1, \hat\alpha_1^2 = \frac{\frac{\alpha_1^2}{\beta_2^1}\alpha_1^3+\frac{\alpha_1^3}{\beta_3^1}\alpha_1^2}{\frac{\alpha_1^2}{\beta_2^1}+\frac{\alpha_1^3}{\beta_3^1}}, \hat\beta_3^2 = \frac{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2+\frac{\beta_3^1}{\alpha_1^3}\beta_5^3}{\frac{\beta_2^1}{\alpha_1^2}+\frac{\beta_3^1}{\alpha_1^3}}, \hat w_3 = \frac{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2 w_4+\frac{\beta_3^1}{\alpha_1^3}\beta_5^3 w_5}{\frac{\beta_2^1}{\alpha_1^2}\beta_4^2+\frac{\beta_3^1}{\alpha_1^3}\beta_5^3}$.

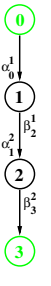

Fig. 3. 2-node Representation

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. □

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

Theorem 3. *We can further aggregate the model in Figure 3 with parameters* $\hat{\alpha}_0^1, \hat{\alpha}_1^2, \hat{\beta}_2^1, \hat{\beta}_3^2, \hat{w}_3$, *into a simpler model as shown in Figure 4 with the equivalent parameters* $\tilde{\alpha}_0^1, \tilde{\beta}_2^1, \tilde{w}_2$ *as follows:* $\tilde{\alpha}_0^1 = \frac{\hat{\alpha}_0^1 \hat{\alpha}_1^2}{\hat{\alpha}_1^2 + \hat{\beta}_2^1}$, $\tilde{\beta}_2^1 = \frac{\hat{\beta}_3^2 \hat{\beta}_2^1}{\hat{\alpha}_1^2 + \hat{\beta}_2^1}$, $\tilde{w}_2 = \hat{w}_3, \tilde{x}_1^* = \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\alpha}_1^2} \hat{x}_1^*$, $\tilde{x}_1^* = \frac{\hat{\alpha}_1^2 + \hat{\beta}_2^1}{\hat{\beta}_2^1} \hat{x}_2^*$.

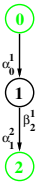

Fig. 4. 1-node Representation

Proof. The proof can be easily carried out by showing that the solution obtained from the optimal solution for one problem is feasible for the other problem, and the two solutions have the same objective value. The details can be found in [11]. □

Besides binary trees, Theorem 2 can also be applied repeatedly to handle general fork trees with arbitrary out-degree(≥ 2). It is straight forward to check formula that result of the merging process in Theorem 2 does not depend on the order of the merging process. It is also straight forward to prove a similar theorem as Theorem 1. The idea of dealing with general tree is to apply Theorem 1 to unit two layer subtree and then replace them with a new node. The following algorithm states the whole process.

Algorithm 2. Backward Shrink Algorithm for Trees
1. If there are 2 leaves with a common predecessor, apply Theorem 2 to these 3 nodes (2 leaves and their predecessor) to find the equivalent 2 node structure. Otherwise, Use Theorem 3 to aggregate the 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 map resource allocation back according to Theorems 2 and 3.

Theorem 4. *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 $O(|\mathcal{E}|)$. The optimality can be proved by induction on the size of the graph. The details are omitted due to limited space. □

4 Distributed Solutions

In this section, we present distributed solutions for the problem. Simulation experiments demonstrate that they perform well even for general network topologies that do not have a tree structure.

4.1 Distributed Algorithms

We develop two heuristics to solve the general problem. These heuristics are based on the 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;)
    - apply Theorem 2 repeatedly to remove one leaf at a time from G;
    - apply Theorem 3 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 1 and 3 to compute solution for *node* and the flow amount to all its children;

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) initialize $(u_1, \ldots, u_k)$ to be $(w_1, \ldots, w_k)$;
1) call Algorithm 1 for the single output problem with $(u_1, \ldots, u_k)$;
2) estimate the gradient for $f(u_1, \ldots, u_k)$;
3) move 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 [13], 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.

4.2 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 $s_i := S_i - \cup_{k \in S_i} S_k$;
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.

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 is 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

Table 1. Results for Heuristic A

# 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 for Heuristic B

# 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

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.

5 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 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 and capable of quickly adapting to real-time fluctuations in the consumption and production rates and changes in resource consumption requirements, while achieving high quality solutions even in non-stationary systems.

References

1. F. Baccelli, Z. Liu. On the Execution of Parallel Programs on Multiprocessor Systems-A Queuing Theory Approach. Journal of the ACM, Vol.37, No.2, April 1990, pp.373-417
2. F. Baccelli, Z. Liu. On the Stability Condition of a Precedence-based Queueing Discipline. *Adv. Appl. Prob.*, Vol 21, 1989, pp. 883-887.
3. F. Baccelli, F. Makowski, and D. Towsley. Acyclic Fork-Join Queueing Networks. *J. ACM*, Vol. 36, 3, 1989, pp. 615-642.
4. C. Baldwin, K.B. Clark, J. Magretta, J.H. Dyer, M. Fisher, D.V. Fites *Harvard Business Review on Managing the Value Chain*, Harvard Business School Press, 2000.
5. A. Brandstdt, V. Bang Le, and J.P. Spinrad. Graph Classes: A Survey. *SIAM*, 1999.

6. E. Coffman and G. Lueker. *Probabilistic Analysis of Packing and Partitioning Algorithms.* Wiley, 1991.
7. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, Second Edition, MIT Press and McGraw-Hill, 2001.
8. A. Mas-Collel,M. Whinston and J. Green. Microeconomic Theory. Oxford University Press, 1995.
9. J. A. Sharp, *Data Flow Computing*, (Ed.), Ablex Publication Corp., 1991.
10. D. Simchi-Levi, P. Kaminsky, E. Simchi-Levi. *Designing and Managing the Supply Chain*, 2 edition, McGraw-Hill/Irwin, 2002.
11. K. Tang, Z. Liu, C. Xia and L. Zhang. Distributed Resource Allocation for Stream Processing Systems. *IBM Research Report.* 2006.
12. S. Viglas and J. Naughton. Rate-Based Query Optimization for Streaming Information Sources. *ACM SIGMOD*, 2002.
13. B. Xi, Z. Liu, M. Raghavachari, C. Xia, and L. Zhang. A Smart Hill-Climbing Algorithm for Application Server Configuration. Proceedings of the 13th International Conference on World Wide Web. 2004, pp. 287-296.