Resource Allocation for Distributed Streaming Applications

Qian Zhu  Gagan Agrawal
Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{zhuq,agrawal}@cse.ohio-state.edu

Abstract—We consider resource allocation for distributed streaming applications running in a grid environment, where continuously streaming data needs to be aggregated and processed to produce output streams. Because such an application comprises a pipeline of processing stages, both communication and computational requirements need to be taken into account while performing resource allocation.

In this paper, we give a rigorous formulation of this resource allocation problem, based on the DAG representation of the application as well as the environment. We have shown how we can use the notion of subgraph isomorphism and developed an effective resource allocation algorithm. The main observations from the experiments we conducted to evaluate our algorithms were as follows: the overhead caused by our algorithm is comparable to an existing algorithm, Streamline, which is based on heuristics. At the same time, the application performance was improved by 30% on average. When compared to the allocation performed by the Optimal algorithm, which enumerates all mappings, the application performance with our algorithm was within 4%. At the same time, unlike the Optimal algorithm, our algorithm scaled well to large graphs.

I. INTRODUCTION

An important development over the last few years has been an emergence of stream model of data (processing). In the stream model of processing, data arrives continuously and needs to be processed in real-time, i.e., the processing rate must match the arrival rate. There are two trends contributing to the emergence of this model for scientific applications. First, scientific simulations and increasing numbers of high precision data collection instruments (e.g., sensors attached to satellites, medical imaging modalities, or environmental sensors) are generating data continuously, and at a high rate. The second is the rapid improvements in the technologies for Wide Area Networking (WAN), as evidenced, for example, by the National Lambda Rail (NLR) and the interconnectivity between the TeraGrid and Extensible Terascale Facility (ETF) sites. As a result, often the data can be transmitted faster than it can be stored or accessed from disks within a cluster. It is often not feasible to store all data for processing at a later time. Also, it is important to react to any abnormal trends or change in parameters quickly. Thus, the analysis needs to be done in real-time or near real-time.

The distributed streaming processing applications we consider are somewhat related to the content distribution applications, which have received much attention lately [11], [19], [20]. However, there is a very important distinction between these two classes of applications. Content distribution applications simply deliver a data stream to a set of hosts. In comparison, the applications we consider perform non-trivial computation, in a pipelined fashion over a series of processing nodes.

We have been developing a middleware and a set of techniques to enable processing of (distributed) data streams over grid resources [10], [9]. The overall goal is to facilitate distributed and real-time scientific stream data processing applications, which can be easily invoked from the web, can flexibly make use of grid resources, and are interoperable, efficient, scalable, and provide quality of service.

One critical problem in distributed data stream processing that needs to be addressed is how to assign resources to streaming applications, which is referred to as the resource matching or the resource allocation problem. Resource allocation is a critical and widely studied problem in grid computing [3], [17], [31]. However, an application processing distributed data streams poses the following significant challenge. Because such an application comprises a pipeline of processing stages, both communication and computational requirements need to be taken into account while performing resource allocation.

Existing solutions to the resource allocation problem for grid streaming applications are mainly based on ad hoc and heuristic methods [5], [17]. Some of the existing solutions have not considered both bandwidth and computing power.

Our solution is based on a well known graph algorithm concept, subgraph isomorphism [24]. Given two graphs, the subgraph isomorphism problem is to decide whether there is a subgraph of one graph that is isomorphic to the other graph. We show how we can model both the streaming application and the underlying environment as directed acyclic graphs (DAGs). We show how a version of our problem, which only considers network bandwidth as the constraint, is a simple modification of the subgraph isomorphism problem. However, we still need to handle computing resources as constraints as well.

Thus, the main contributions of our work are as follows.

- We have given a rigorous formulation of the resource allocation problem for distributed streaming applications, based on DAG representation of the application as well as the environment, and a simple effectiveness value, which captures the quality of any candidate mapping.
- We have shown how we can use the notion of subgraph isomorphism and modify an existing algorithm to address a version of our problem which only considers network bandwidth as the constraint.
- Next, we show how computing power can be considered
together with network bandwidth, by a simple modification of the framework.

Both a streaming visualization application and a synthetic application have been used to carefully evaluate our algorithms. The main observations from our experiments were as follows:

- The overhead caused by our algorithm is comparable to an existing algorithm, Streamline [5], which is based on heuristics. At the same time, the application performance was improved by 30% on average.
- When compared to the allocation performed by the Optimal algorithm, which enumerates all mappings, the application performance with our algorithm was within 4%. At the same time, unlike the Optimal algorithm, our algorithm scaled well to large graphs.

The rest of the paper is organized as follows. We formulate the resource allocation problem and formally define subgraph isomorphism in Section 2. The detailed resource allocation algorithm is described in Section 3. Section 4 reports the results from experimental evaluation. We compare our work with related research efforts in Section 5 and conclude in Section 6.

II. RESOURCE ALLOCATION IN DISTRIBUTED DATA STREAM PROCESSING

This section formally describes the resource allocation problem for distributed data stream processing in a heterogeneous grid computing environment. We first define a general processing model for distributed processing of data streams. Throughout this paper, we refer to both the bandwidth of the network links and the computing power of the processing nodes as the resources.

A. Problem Statement

In data stream processing applications, data arrives continuously from distributed sources. We formulate a general data stream processing model as a Directed Acyclic Graph (DAG). In this model, each processing unit may require multiple input data streams simultaneously and produce one or more output streams. A round-robin scheduling policy is assumed for allocating the network link to multiple data transfers originating at the same node. It is also assumed that a processing unit starts transferring data only after it has completed processing the current input. Its successor node starts execution on a particular data packet as soon as it is received.

The DAG representing the data stream processing model comprises of sources, sinks, and processing nodes, with directed edges denoting the flow of data between the nodes. The sources, which only have out-going edges, generate data at a certain rate, while the sinks, which only have incoming edges, output results to the users. A processing node represents a processing unit, which receives data from one or more sources or upstream processing nodes, processes the data, and transfers intermediate results to one or more downstream processing nodes or sinks. Typically, the data stream from a source is processed by a pipeline of processing units. Such a chain of producer-consumer pairs is referred to as a path. The DAG representation of the streaming application is denoted as $G_P(V_P, E_P)$, where each vertex $v^p_j$, $v^p_i \in V_P$ represents a source, a sink, or a processing node and is associated with a computing power requirement, denoted as $W^p_i$. A data transfer between node $i$ and $j$ is represented by an edge $e^p_{ij}$, $e^p_{ij} \in E_P$. There is a bandwidth requirement, denoted as $B^p_{ij}$, along each edge specified by the user. An example of $G_P$ is shown in Figure 1(a). In this example, the user specifies $W^p_1 = 800$, meaning that node 1 should be able to process 800k data per second. Also, the user requires $B^p_{1,2}$, the bandwidth between node 1 and 2, to be 100k/sec.

The underlying network is represented as a graph $G_R(V_R, E_R)$ with vertices in $V_R$ representing the actual physical nodes, and the edges in $E_R$ representing the network connections between nodes. We use $W^R_i$ to denote the computing power of node $v^R_i$ that is the amount of data(in kilobytes) can be processed per second. We further associate with each edge $e^R_{ij}$ a weight $B^R_{ij}$ to denote the network bandwidth of the corresponding link. Figure 1(b) illustrates an example of $G_R$.

In general, both the processing power and the network bandwidth could be important considerations in resource allocation. Since all the sources and sinks in the application model($G_P$) are placed at fixed locations in the underlying network ($G_R$), we view this problem as that of a mapping $M$ from the processing nodes in the model to the network graph, namely, from $G_P$ to $G_R$. More specifically, this mapping assigns each node, $v^p_i \in V_P$ to a node, $v^R_i \in V_R$. Thus, $M$ implies a corresponding mapping of edges in $G_P$ to edges in $G_R$, such that each edge between connected processing nodes $v^p_i$ and $v^p_j$ is mapped to one network path with the possible highest bandwidth between the nodes that $v^p_i$ and $v^p_j$ are assigned to. Clearly, a large number of mappings are possible. Our goal is to determine the optimal mapping, i.e. the one which is able to achieve the maximum throughput from the application by making the best resource usage. For long running streaming applications, achieving maximum throughput can be the same as achieving the minimum execution time, since operations during the start-up and the end can be ignored.

B. Subgraph Isomorphism

The mapping problem we have described above is very similar to the classical subgraph isomorphism problem which has been studied in graph theory [24].

Definition A graph $G_1 = \{V_1, E_1\}$ is isomorphic to a subgraph of $G_2 = \{V_2, E_2\}$, denoted by $G_1 \cong G'_2 \subseteq G_2$, where $G'_2 = \{V'_2, E'_2\}$, if there is an injection $\varphi : V_1 \rightarrow V'_2$, such that for every pair of vertices $v_i, v_j \in V_1$, if $(v_i, v_j) \in E_1$ then $(\varphi(v_i), \varphi(v_j)) \in E'_2$.

Subgraph isomorphism is a decision problem that is known to be NP-complete[16]. We show the subgraph isomorphism mapping through an example, as presented in Figure 1(a) and (b). There are six subgraphs of $G_R$ that are isomorphic to $G_P$. One mapping $M$ is $\{(1, A), (2, B), (3, C), (4, E)\}$, where $(1, A)$ denotes that the node 1 in $G_P$ is mapped to the node $A$ in $G_R$.

Thus, the resource allocation problem in our context becomes that of choosing an isomorphic subgraph from $G_R$. 
However, as the example above shows, there could be a large
count of such subgraphs. Thus, we need to be able to search
through the set of isomorphic subgraphs, and choose the best
one. For this purpose, we associate every mapping with an
effectiveness value, denoted as $Eff(M)$, which captures how
effective the resource allocation generated by this mapping
would be. The main idea here is that being unable to meet the
resource requirement will cause a penalty, and thus would
increase the value of $Eff(M)$. The formal details of calculating
$Eff(M)$ are discussed later in this section.

However, the actual problem we need to consider is not
precisely the exact subgraph isomorphism problem. This is
because in a situation where direct connection between two
nodes in the network has a low bandwidth, a better resource
allocation scheme may choose a path connecting these two
nodes with high bandwidth. Therefore, we use modified
subgraph matching in our approach. The purpose is to find a
subgraph of $G_R$ that can be isomorphic to $G_P$ after a series
of node deletion operations.

**Definition.** A modified subgraph isomorphism from a graph
$G_1$ to a graph $G_2$ is a bijective function $f : v_1 \leftrightarrow v_2$, where
$v_1 \in V_1, v_2 \in V_2'$ and $(v_1', v_2') \in E_1$ is mapped to
$(f(v_1'), f(v_2')) \in E_2'$.

We use the example presented in Figure 1(a) and (b) to show the
modified subgraph isomorphism we defined above. The
mapping $\{(1, A), (2, B), (3, C), (4, D')\}$ is a match with the
above definition of modified subgraph isomorphism. Here,
the edge $3 \rightarrow 4$ of $G_P$ is not directly mapped to the edge $C \rightarrow\nE$ in $G_R$. Instead, it is matched with a simple path $C \rightarrow D \rightarrow\nE$ after node $D$ is deleted. In general, vertex deletion means
deleting nodes along a path in $G_R$ before it can be mapped to
a single edge in $G_P$. In our scheme, instead of deleting these
nodes, we use them as transporters. We denote $c_{vd}$ as the
cost overhead with a modified subgraph isomorphism, which
captures the overhead of adding such transporters. The cost
$c_{vd}$ is further added in calculating the effectiveness value of a
mapping.

**C. Performance Goal**

Before finalizing the problem formulation, we describe the
performance goal of our defined resource allocation problem.
The problem we are considering involves finding an initial
static mapping of the application DAG onto the underlying
network to minimize the overall execution time. Alternatively,
we can achieve the goal by minimizing the end-to-end latency
of the bottleneck path, i.e., the processing chain that takes
the longest execution time. We denote $T_{comp}(v_R')$ as the
computation time on node $v_R'$ and $T_{comm}(v_R', v_R')$ is used to
denote the communication time from node $v_R'$ to $v_R$. Thus, we
can define our performance goal as to minimize the following
expression:

$$\max_{v'_k} \sum_{v'_R, v'_R} \left( T_{comp}(v'_R) + T_{comm}(v'_R, v'_R) + T_{penalty} \right)$$

(1)

with constraints

$$B_{P}^{v'_k} \leq B_{R}^{v'_R}$$

(2)

$$W_{P}^{v'_k} \leq W_{R}^{v'_R}$$

(3)

where $p_k$ denotes any path in the processing model and $T_{penalty}$ is the overhead of adding transporters. The two
constraints specify that the bandwidth and computing power
requirements from the processing model, $G_P$, should be
satisfied by the allocated resource under the current mapping.
Recall that a node $v'_k$ is assigned to $v'_R$.

Let us first assume that the only constraint is the bandwidth,
i.e., $T_{comp}(v'_R) = 0$ in Formula 1. Given the mappings from
$G_P$ to subgraphs of $G_R$, we can calculate the application
execution time using the Formula 1 and chooses the one with
the lowest execution time. However, this could be very time
consuming if the number of candidate mappings is large. In
our implementation, we used an Effectiveness value associated
with each mapping, representing how effective it is in terms of
resource usage. More specifically, we first estimate the
communication time $T_c(v'_P, v'_P)$ for each link in $G_P$ and then
compare it with the communication time on the links that $G_P$ has
mapped onto. Thus, we see the performance gain we can get
from current mapping.

Formally, we use a function $F$ which is based on $B_{Q}^{v'_k, v'_R}$ from
$G_P$ and $B_{R}^{v'_k, v'_R}$ from $G_R$. As a higher bandwidth results in
lower communication time, an increase in the value of the
function $F$ represents a decrease in the execution time with
the current mapping. The Effectiveness value of each edge is
computed in evaluating $F$, giving credit to the edges which meet
the requirement and penalizing the ones that do not.
Furthermore, more credit will be given to edges with higher
bandwidth. Similarly, edges suffering with lower bandwidth
will be penalized more. As we discussed before, adding in transporter nodes will also cause a penalty. The effectiveness value of the mapping, referred to as $Eff(M)$, is the sum of the effectiveness value for each edge.

In our work, we apply the tangent sigmoid function $[1]$ to be the function $F$, which is a non-linear transfer function, to allow the computed value to be within a specified range $(-1, 1)$. If the bandwidth of the edge in $G_R$ satisfies the requirement in $G_P$, the tangent sigmoid function produces a positive value, otherwise, its output is negative.

Now, we formulate our performance goal as

$$Eff(M) = \sum_{v'_p,v'_p \in V_P, v''_R,v''_R \in V_R} F(B^i_P, B^j_R) + \sum_{k_{vd}} c_{vd} \tag{4}$$

where $m$ is the number of transporters in the mapping. In our implementation, we assigned $c_{vd} = -0.02$.

Next, we include the computing power as another resource constraint. We further add another expression, taking $W^t_P$ and $W^t_R$ as the input.

$$Eff(M) = \sum_{v'_p,v'_p \in V_P, v''_R,v''_R \in V_R} F(B^i_P, B^j_R) + \sum_{k_{vd}} c_{vd} + \sum_{v'_p \in V_P,v''_R \in V_R} F(W^t_P, W^t_R) \tag{5}$$

III. ALGORITHM DETAILS

We now describe our resource allocation algorithm. Initially, we present an algorithm considering the bandwidth of network connections as the only constraint. Then, we show how the computing power of nodes can also be taken into account by just modifying the problem formulation.

The subgraph isomorphism algorithm we base our work on is the VF $[22], [23]$ algorithm proposed by L.P. Cordella et al. The VF algorithm provides the solution to the subgraph isomorphism problem, so, we had to modify it for the modified subgraph isomorphism. Initially, we review the VF algorithm as developed by Cordella et al.

A. Background: VF Algorithm

In this algorithm, a mapping $M$ is expressed as a set of ordered pairs $(v'_p, v''_R)$, with $(v'_p \in G_P$ and $v''_R \in G_R)$, each representing the match of a node $v'_p$ in $G_P$ and a node $v''_R$ in $G_R$. The graph matching process is efficiently described using the State Space Representation (SSR). Each state $s$ of the matching process represents a partial mapping solution, which is denoted as $M(s)$. The state of a complete mapping is denoted as $s_f$ and a complete mapping is denoted as $M(s_f)$. We use $E_P(s)$ and $V_P(s)$ to denote the edges and nodes from $G_P$ which are involved in current partial solution. Similarly, $E_R(s)$ and $V_R(s)$ are used for $G_R$.

The main issue is to reduce the number of paths to be explored during the search. For each state on the path from the initial state $s_0$ to a goal state $s_f$, the algorithm tests the partial solution to verify certain coherence conditions. To be specific, a node pair $(v'_p, v''_R)$ is valid and can be added into $M(s)$ if it satisfies the certain feasibility rules, as proposed in the paper $[22], [23]$. We want to be able to see if a state $s$ has no successors after a certain number of steps. The feasibility rules can help detect conditions leading to incoherence as soon as possible and the states violating the rules will be discarded for further expansions.

Algorithm III.1: RESOURCE ALLOCATION($G_P, G_R$)

ModifiedSubgraphIsomorphism(s)

INPUT an intermediate state $s$

$s_0$: the initial state, $M(s_0) = \emptyset, Eff(s) = 0$

$M_k = \emptyset$, $M_f = \emptyset$

OUTPUT $M_E$: the most efficient mapping from $G_P$ to $G_R$

if $s = \emptyset$

then buildCandidateLists();

while (true)

if $M(s)$ covers all the nodes in $G_P$

then

$M_f = M_f \cup (M(s))$;

$M_p = M_p - \{M(s)\}$;

else updateCandidateList();

// generate all the possible match pairs

$p(s) = possibleMatch(s)$

for each $p = (v'_p, v''_R) \in P(s), v'_p \in V_P(s), v''_R \in V_R(s)$

for each $M_k(s) \in M_p$

if $(VF_feasibilityCheck(p, M_k(s)) == true)$

then

// compute the state $s'$ obtained by adding $p$ to $M_k(s)$

$s' \leftarrow VF_updateState(s, p)$;

// $ex$ are the newly added edges

$Eff(M_k(s')) = Eff(M_k(s)) + Eff(ex)$

MODIFIEDSUBGRAPHISOMORPHISM($s'$);

if conflicts found in $M_k(s)$

then $M_p = M_p - \{M_k(s)\}$;

if $M_p = \emptyset$

then $Eff(M_E) = max\{Eff(M_k(s)), M_k(s) \in M_f\}$

break

return ($M_E$)

Fig. 2. Static Resource Allocation Algorithm

The algorithm proceeds as a depth-first search. After a new node pair $(v'_p, v''_R)$ is added into the partial mapping solution, the algorithm generates all the candidate pairs composed of all the nodes adjacent to $v'_p$ through its outgoing edges and those of node $v''_R$.

B. Algorithm for Static Resource Allocation

The resource allocation algorithm using modified subgraph isomorphism is shown in Figure 2. We first introduce some notation before presenting our algorithm. We define a candidate node list for each node $v'_p$ in $G_P$, denoted by $Cand(v'_p)$. It records the possible nodes in $G_R$ that $v''_R$ can be mapped to. We consider a node $v''_R$ in $G_R$ to be a candidate for a node $v'_p$ in $G_P$ if its degree of both outgoing and incoming edges are greater than or equal to those of $v'_p$. We assume that $Cand(v'_p) \neq \emptyset$.

In our running example, $Cand(3) = \{C, D, G\}$.

Our modified subgraph isomorphism algorithm begins with building such candidate node lists. We traverse $G_R$ in a depth-first order until all the candidates for each node $v'_p$ in $G_P$ are found. The function buildCandidateLists() also links nodes to form a sublist in each $Cand(v'_p)$, if they are connected in graph $G_R$. This facilitates the modified match as we will explain later. In the example shown in Figure 1(c), we have a sublist as node $C$ pointing to node $D$ in $Cand(3)$, since they are connected in $G_R$. Then, the possibleMatch() generates
all the possible match pairs for \( v_p \) as a start, based on the candidate node lists from the previous step. Compared to the original VF algorithm, we reduce the redundancy in candidate node pair generation. For example, if we pick \( v_p \) to be node 1 and add in \((1, A)\) to the current partial mapping, we have to check a total number of 15 pairs in the original VF algorithm, as node \( A \) has five outgoing edges and node 1 has three. However, as we can see, none of nodes \( B, E, F \) is in the \( Cand(3) \), that means that we do not need to check the pair \((3, B)\), \((3, E)\), or \((3, F)\). Similarly, we should skip the pairs \((4, B)\), \((4, C)\), and \((4, G)\) since \( Cand(4) \) does not include node \( B, C, \) or \( G \). In this way, we reduce the number of pairs needed to be checked down to 9.

Furthermore, as we allow vertex deletion for modified subgraph isomorphism, the algorithm also produces possible match pairs involving the nodes in a sublist if we choose a node from the same sublist in another possible match pair. The reason is that the extra nodes along the path could be deleted. As shown in the example, after we generate the candidate pair \((3, C)\), we also generate the pair \((3, D)\), with node \( C \) working as a transporter.

In terms of discovering multiple mappings, we propose a Multiple Partial Mapping Set, denoted as \( M_p \), in Figure 2. The completed mapping set is denoted as \( M_f \) in the algorithm. After generating all the possible candidate node pairs, the algorithm proceeds with generating new partial mappings, if current mapping state is \( \emptyset \), or updates existed ones with each of the possible match pairs, as we call the VF subroutine \( VF_{updateState}() \). We apply the feasibility rules proposed in the VF algorithm to check whether a potential match pair could be included in any of the partial mappings, by invoking the VF subroutine \( VF_{feasibilityCheck}() \). If so, we update the current mapping state by extending it with the newly added pair and calculate the effectiveness value associated with current mapping. Finally, the algorithm recursively calls the ModifiedSubgraphIsomorphism function with the new state until a complete mapping is found. Then we add the successful mapping to the set \( M_f \) and delete it from \( M_p \).

If the set \( M_p \) is empty, i.e., no more mappings can be found, we report the mapping with the highest effectiveness value, calculated using the Formula 4. Note that during the entire procedure, a partial mapping will be discarded if a conflict is detected.

C. Handling Computing Power as a Constraint

We now discuss how our resource allocation algorithm can also take into account the computing power of each processing node.

We use a simple modification to our formulation, which allows the subgraph isomorphism based approach to be applicable without causing extra time or space complexity. We view the processing of data on a node as data going through a network link with a limited bandwidth. In other words, we treat each computing node as a network link. The amount of data that can be processed within a unit time, which represents the computing power on this node, is converted to bandwidth capacity. After this conversion, we can apply the previous algorithm by adding in the effectiveness value on each processing node. Now we can handle a situation where both bandwidth and computing power are considered as resources. The Formula 5 defined earlier is used as the performance goal.

However, there could be cases in which a high bandwidth network link connects to a node with a low computing power, or, a high computing power node is connected through a low bandwidth link. Given a choice, we favor the first case in our resource allocation. The reason is the following. When the communication bandwidth is limited, some nodes will partially starve. These nodes will have some idle time due to the slow communication rates, no matter what their processing speeds are. However, a high bandwidth and a correspondingly low computing power could cause the problem that data cannot be processed in time. This can lead to the requirement of a very large input buffer size at the node. One possible solution could that instead of one-to-one node mapping, we allow vertex insertion operations, i.e., extra nodes in \( G_R \) are considered to match with one node in \( G_F \). In this way, we can map one node in the processing model to a cluster of actual network nodes.

IV. EXPERIMENTAL EVALUATION

This section presents results from a number of experiments we conducted to evaluate the resource allocation algorithm we have presented in this paper. For our evaluation, we focused on two alternative resource allocation approaches: Optimal chooses the best resource assignment with the lowest application execution time based on an exhaustive search over all possible allocations. It gives us a baseline for comparison. Streamline [5] applies heuristics in scheduling streaming applications on the grid. It prioritizes the nodes in the application according to their computation and communication requirements and proceeds by meeting the requirements of most needy stage first.

Specifically, we had the following goals in our experiments:

- Evaluate the scalability of our resource allocation algorithm, with increasing size of the graphs (Section IV-B).
- We show how our algorithm has similar execution time as Streamline, and unlike Optimal, it is suitable for large graphs.
- Demonstrate that our algorithm produces mappings which result in high performance from applications (Section IV-C). Particularly, we show that the results from our algorithm are quite comparable with Optimal, and significantly better than Streamline.

A. Experiment Setup

For distributed processing of streaming data in a grid environment, we need high bandwidth networks. However, for our study, we did not have access to a wide-area network that gave high bandwidth and allowed repeatable experiments. Therefore, all our experiments were conducted within a single Linux cluster, with bandwidth being synthetically limited and network topology being randomly constructed. The cluster consists of 64 computing nodes. Each node has 2 Opteron 250 (2.4GHz) processors with 8GB of main memory and 500GB local disk space, interconnected with switched 1 Gb/s Ethernet. We set up an environment in which network bandwidth was...
simulated by inserting delays between packets. The range of bandwidth variance was from 100k/s to 1Gb/s and the actual end-to-end bandwidth was obtained from Network Weather Service (NWS) [27]. In this way, we were able to demonstrate that our resource allocation algorithm is effective when network bandwidth can vary significantly and have severe impact on the communication time. We introduced a synthetic value of computing power from a non-uniform distribution for each of the physical nodes. Similarly, the computing power varies significantly enough to impact the computation time.

The experiments were conducted using a volume rendering application [14] and a synthetic application. The goal of the volume rendering application is to create a 2D projection of a 3D data set (volume data). We consider a situation where data collected from sensors or that generated by long running simulations needs to be visualized in real-time. Because of the resources required for rendering, and the physical distance between the source producing the data and the display, a distributed pipeline of processing stages is required. Because visualization operations are compute-intensive and the volume of data to be communicated can be very large, assigning resources carefully to each processing unit and data transfer link in a grid environment is critical.

We also used a synthetic application for experiments. In this application, we randomly constructed a DAG as the processing model and transferred data between nodes. On each node, we simulated the processing by keeping the data on the node with a certain delay. We made the time spent on communication dominate the whole procedure to show the importance of making good use of limited bandwidth resources. The details of the DAGs for both volume rendering and synthetic applications are shown in Figure 3. We used edge density, denoted as $\eta$, to be the probability of an edge between any two given nodes. The number of vertices in the synthetic application was chosen as 5, 10, and 15.

### B. Scalability of the Resource Allocation Algorithm

We first compared the overhead caused by our resource allocation algorithm to other algorithms, with increasing number of vertices the graphs. For this experiment, we randomly constructed an application graph ($G_P$) with 10 nodes and 60% edge density. The environment or network graph ($G_R$) was chosen from the graph-base used for evaluations for the VF algorithm [23]. We varied the number of vertices from 20, 40, 80, 100, 200 to 400, and set $\eta$ to be 0.6.

The distribution of execution time for different resource allocation algorithms is presented in Figure 4. The Optimal algorithm took 77.53 seconds to complete the mapping procedure when the number of nodes in $G_R$ is 20. It dramatically increased to 1 hour and 50 minutes when $G_R$ has 40 nodes. As it became clear that it is not scalable, we did not execute it for larger graphs. We observed that our algorithm performed comparably to Streamline. Overall, this experiment shows that our algorithm is scalable. Furthermore, the case with 400-vertex graph only took 28.89 seconds to complete. For long running streaming applications, such overhead is clearly acceptable.

### C. Performance of Static Resource Allocation

We used both the volume rendering and the synthetic applications for this experiment. We compared the execution time, excluding the overhead of the resource allocation algorithms, when applications ran under different resource configurations generated by the three algorithms. First, we show the performance of the volume rendering application. We varied the error tolerance, denoted as $\tau$, in our experiments. A small value of error tolerance implies more communication as well as computation. The results are presented in Figure 5. As we can see, the performance of Optimal and our algorithms
are comparable. In fact, they are within 4% for the case with highest workload, i.e., when $\tau = 0.000$. Compared to Streamline, our algorithm results in an improvement of 33%, 29%, and 27%, with $\tau$ being 0.000, 0.007, 0.1, respectively. These results are to be expected, as we are using a rigorous formulation, whereas Streamline is based on simple heuristics. More specifically, Streamline first assigns most of the available resources to the processing path with the highest computation and communication requirements, leading to the possibility that the resource requirements from other paths are not met. As a result, any of these other paths could end up with a longer execution time than that of the path which was assigned resources first.

We also carried out this experiment with the synthetic application. As demonstrated in the Figure 6, the performance of our algorithm is very close to that of the Optimal algorithm (within 3% for all three cases). The improvement over Streamline was 40%, 36% and 34%, respectively, with different $\tau$ values.

V. RELATED WORK

We now discuss related work in solving resource allocation problem in distributed data stream processing, and more generally, for grid computing. We also list prominent efforts on developing and applying subgraph isomorphism algorithms.

Resource Allocation for Stream Processing: Allocation of processing and communication resources has been an active topic in distributed data stream processing systems [6], [29], [12]. STREAM [6] and Telegraph [29] mainly focus on static and dynamic operator scheduling, providing near-optimal resource allocation strategy. Our work considers a different problem, i.e., we focus on assigning resources to distributed streaming applications, while most of the literature cited above addresses the scheduling of applications with an assumed allocation scheme. Moreover, we consider the resource allocation problem in a grid computing environment, i.e., in a distributed system that crosses multiple administrative domains.

Our use of a directed acyclic graph (DAG) as the application model is similar to the work of Tang et al. [4] and Ali et al. [28]. Tang et al. solve the CPU resource allocation problem in stream processing systems with the objective of maximizing the total return of multiple output streams. Our work consider both bandwidth and computing power, maximize the allowable increase in system workload.

Other Work on Resource Allocation for Grid Computing: Resource allocation has been an important topic in the grid community. Most of the initial work has been on static matching of the resource requirements and the available resources [3], [17], [31]. Hong et al. [7] and Beaumont et al. consider bandwidth-aware resource allocation in heterogeneous computing environment. They solve the system throughput maximization problem with a linear programming formulation. However, they both assume that all the tasks reside on a single node initially, while our work considers a different model, i.e., input streams come to the system from different locations and the results are sent to multiple output nodes.

Subgraph Isomorphism Algorithms and Applications: (Sub)graph isomorphism has been well studied in graph theory [30], [13], [2], [8], [23]. In this paper, we extend the idea used in the VF [22], [23] for the modified subgraph isomorphism defined in our work. Subgraph isomorphism algorithms have been applied to substructure search in cheminformatics/bioinformatics [25], VLSI design [21], similarity measures for structured representations [18], [26], mobile robot design [15], among others.

VI. CONCLUSIONS

This paper has considered the resource allocation problem in distributed streaming applications running in a grid environment, where continuously streaming data needs to be aggregated and processed to produce output streams. We considered a directed acyclic graph (DAG) to model the processing of data streams. We have proposed a modified subgraph isomorphism algorithm to address the resource allocation problem, with the goal of maximizing the throughput of the system. Though our initial formulation considers only the network bandwidth as the constraint, we show how computing power can be considered by a simple modification of the approach.

The main observations from the experiments we conducted to evaluate our algorithms were as follows: the overhead caused by our algorithm is comparable to an existing algorithm, Streamline, which is based on heuristics. At the same time, the application performance was improved by 30% on average. When compared to the allocation performed by the Optimal algorithm, which enumerates all mappings, the application performance with our algorithm was within 4%. At the same time, unlike the Optimal algorithm, our algorithm scaled well to large graphs.

ACKNOWLEDGEMENTS

This work was supported by NSF grants 0541058 and 0619041. The equipment used for the experiments reported here was purchased under the grant 0403342.

REFERENCES


