Cost Estimation of Parallel Constrained Producer-Consumer Algorithms

Tariq Kamal[1,2], Keith R. Bisset[2], Ali R. Butt[1], Madhav Marathe[1,2]

[1]Department of Computer Science, Virginia Tech, Blacksburg, VA 24061 USA
[2]Virginia Bioinformatics Institute, Virginia Tech, Blacksburg, VA 24061 USA
Email: [1]{butta}@cs.vt.edu, [2]{tkamal, kbisset, mmarathe}@vbi.vt.edu

***Abstract*—Cost estimation is crucial in the performance modeling of parallel algorithms and allocation of computational resources on distributed systems. This paper presents a novel methodology for estimating the cost of constrained producer-consumer (CPC) algorithms. In CPC algorithms, the computation is performed by classes of nodes (tasks), separated in time. The methodology combines data flow analysis with communication latencies to determine the production and consumption of data on different processors, which helps in determining the amount of computations and communication. The cost metric that we develop in this paper uses computational imbalances and communication load, and determines a single cost value. The resulting metric is unique, as it provides the first model that targets CPC algorithms. It has wide application in Genetic Algorithms, molecular dynamics, scheduling schemes and computational epidemiology. We provide a general method for determining the application-specific constants of the cost metric. As an example, we extract the constants for EpiSimdemics (a highly scalable contagion simulator), and give guidelines for applying the procedure to other CPC algorithms. Our evaluations show that the cost metric estimated the execution times of a contagion simulator with less than a 6.5% error. The metric can be used in optimal assignment of computational resources.**

***Keywords*-Cost Estimation; Producer-Consumer Parallel algorithms; Distributed Systems; Resource Allocation;**

I. Introduction

Producer-consumer-based modeling is a natural and widely used approach for modeling complex systems composed of multiple interacting entities. A large group of parallel algorithms belonging to the class of producer-consumer models can be classified as constrained producer-consumer (CPC) algorithms, also called bulk synchronous parallel (BSP) [1], [2] model. Processing of a CPC parallel program on a distributed system is performed iteratively by classes of tasks, (C_1,...,C_N), in *N* phases. The tasks in successive classes have a producer-consumer relationship. In the first *N*-1 phases, tasks in C_k produce messages for tasks in C_{k+1} to consume. In the last phase (phase *N*), tasks in C_N produce messages for tasks in C_1 to consume. One distinguishing feature of these types of algorithms is that the processing of consumer tasks will not start until they receive all the messages destined for them.

A large number of applications belong to the class of CPC algorithms. They range from modeling Genetic Algorithms [3], [4] and molecular dynamics [5], to simulating the outbreak of epidemics [6]–[8] and threat of mobile malware [9], from modeling the human immune system [10] to understanding the consumer purchasing behavior [11], and many others. Besides applications, many frameworks naturally fit into the CPC category. One such example is iterative mapreduce [12], [13], where the program is run iteratively by sets of mappers and reducers. In iterative mapreduce, the mappers and reducers work as producers and consumers, alternatively.

Given that a considerable number of applications are modeled using CPC algorithms, we posit that it is worthwhile to explore the load characteristics of such applications aimed at estimating their execution cost. The cost metric will help in evaluating various data partitioning and load balancing schemes, estimating the execution cost of a study, or a series of studies and in the allocation of computational resources. Moreover, a number of future applications, including load balancing schemes and performance analysis tools, can benefit from it.

An interesting feature of many CPC algorithms is that the computations performed by its tasks are a function of the number of messages exchanged between them. In each iteration, the producer needs to send data to a subset of the consumers in the form of messages, where, the consumers perform computations on received messages. Given the semantics of the simulation algorithm and data-flow analysis, we can determine the number of computations performed by each task.

In this paper, we present a novel methodology for estimating the cost of CPC algorithms. Our work combines data flow analysis with estimation of computational imbalances and communication load to develop a cost metric, which estimates the execution cost of a CPC parallel program. The cost metric has *N*+1 components: *N* computational components (one for each class) and one communication component. The execution cost of a class (of tasks) is determined by the most compute-heavy processor (in the presence of synchronization, as shown in Figure 1), therefore, we use the load of the most loaded partition (of each class) to quantify its computational load. The communication load is measured in the number of messages exchanged between processors, referred to as remote messages. We use only one component to represent the communication load in the cost metric. The regression method (that we use in Section V to fit the model and determine their constants) will adjust the communication components and their constants according to the communication latencies in the target

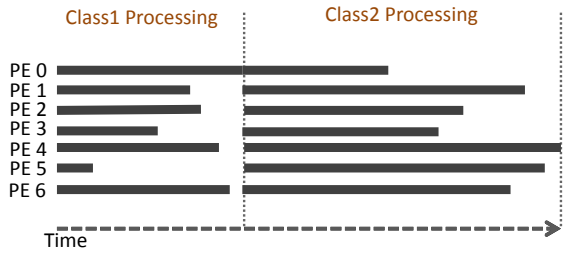

Fig. 1. The execution times of two classes ($Class_1$ and $Class_2$) of tasks on seven processors. Due to synchronization, the most loaded processors, PE0 during $Class_1$ execution and PE4 during $Class_2$ execution, make other processors wait for them to reach synchronization.

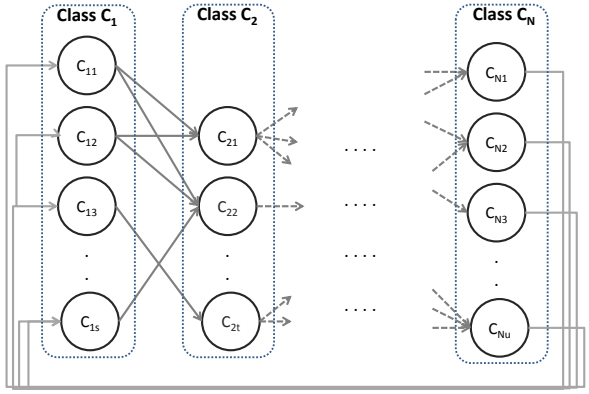

Fig. 2. Entities in CPC parallel-program are shown by a multi-partite interaction graph. The nodes represent the tasks, and the edges show data dependency between the tasks.

application. The assignment of tasks to partitions plays a vital role in the total cost of simulation. We assume the assignment of one partition (of every class) to each computational element (e.g., core). But, the model is equally applicable when assigning multiple partitions of a class to a processor(s).

As an example, we tested our cost metric for EpiSimdemics [6], [7], [14], a scalable parallel simulator that models the spread of contagion in a population. Appropriate coefficients for the three cost metric components (two computational components and one communication component) are derived using regression modeling and execution times from running a variety of data-sets. The cost metric demonstrates the benefits in optimal scheduling and allocation of computational resources (see Section VII). Moreover, it compares the performance of partitioning algorithms. We illustrate the method for EpiSimdemics; however, the methodology is equally applicable to the broad class of CPC parallel applications. The complexity of the model increases linearly with an increase in the number of classes.

Although our definition of CPC model is similar to BSP, we have some differences/additions to it. First, in CPC, the computational load of tasks in a class is related to the amount of incoming and outgoing messages. Second, CPC models are iterative, where each iteration has N phases. Third, the cost of the algorithm is not necessarily linear, and may be a combination of linear and non-linear terms. Fourth, we show a detailed statistical modeling to extract the cost equation and constants associated with its components. Finally, we capture the computation/communication overlap using statistical modeling.

In particular, our key contributions are:

- First study of computational and communication loads in the Constrained Producer-Consumer parallel algorithms,
- A cost metric for the cost estimation of CPC algorithms,
- Methodological determination of the cost metric constants for EpiSimdemics using statistical regression analysis, and
- Demonstration of the benefits of the cost metric in performance evaluation and allocation of computational resources on a distributed system.

The rest of the paper is structured as follows. Section II formulates a general form of CPC algorithm. Section III performs load analysis of CPC algorithms. Section IV develops a metric for the cost estimation of CPC algorithms. Section V develops a methodology for modeling fitting and determination of constants. Section VI determines the cost metric for two example applications (EpiSimdemics and ASPARAGOS). Section VII demonstrates the benefits of the cost metric in performance evaluation and allocation of computational resources. Section VIII compares this work to the related state-of-the-art and Section IX presents concluding remarks.

II. The Constrained Producer-Consumer Algorithm

In this section, we show a generalized form of the CPC parallel-algorithm. Entities in the model can be characterized by a multi-partite interaction-graph as shown in Figure 2. The vertices represent the tasks, and the directed edges correspond to communication from one task to another.

Based on the type of computation they perform the tasks are divided into *N* distinct sets of nodes. Tasks in successive classes act as producers and consumers. In each step, the consumers ensure that they have received all the messages sent to them before starting to process them.

1: initialize(); ▷ assign tasks to processors
2: **for** time-step $= 1$ **to** termination condition **do**
3: **for** k $= 1$ **to** N **do**
4: **foreach** $c \in C_k$ **do**
5: $MO_c \leftarrow$compute(c); ▷ process received messages and generate new messages
6: sendMessages(MO_c); ▷ send messages
7: **end for**
8: **foreach** $c \in \begin{cases} C_1 & \text{if } k = N; \\ C_{k+1}, & \text{otherwise} \end{cases}$ **do**
9: $MI_c \leftarrow$receiveMessages();
10: compute new task state
11: **end for**
12: synchronize(); ▷ ensure all msgs received
13: **end for**
14: **end for**

Fig. 3. The general CPC parallel-algorithm, where c represents a task. MO_c refers to the outgoing messages of task c, MI_c refers to the incoming messages of task c, and N refers to the total number of phases. The termination condition can refer to a specific number of time-steps or a more complex convergence criterion.

The algorithm progresses through several time-steps. A time-step is divided into *N* phases, one per class.

1) At the start of each phase k, tasks in C_k generate and send messages. The message has a destination task in C_{k+1} if $k<N$. Otherwise the message is destined for a task in C_1.
2) After all the tasks in C_k have finished sending their messages, a synchronization happens. This is a global synchronization, guaranteeing that all messages have been received by all the consumers.
3) The consumer tasks process their received messages, and update their state before starting the next phase.
4) When tasks in C_1 have received their messages from tasks in C_N, the next time-step starts.

III. Load Analysis and Characterization

In this section, we perform load analysis of the general CPC algorithm discussed in Section II. The parallel CPC program can be characterized by an interaction graph, G($C_1, \ldots, C_N, E$), as shown in Figure 2. The vertices represent the tasks, and the edges correspond to communication dependencies between those tasks. The weight of a vertex gives the relative computational load of that task, and the weight of an edge gives the relative amount of communication required between the tasks. Vertices (tasks) are grouped into *N* distinct classes $(C_1, \ldots, C_N)$. Each task works as a producer and a consumer, alternatively, and is assigned to exactly one class.

An important feature of CPC algorithms is the communication and its relationship with computation. The quantitative relationship of the data received on a consumer task, and the computation performed by it is application-specific. However, the computation performed by a task in a CPC application is related to the number of consumed and produced messages. For determination of the number of computations of a task, the number of messages generated or received by it needs to be known.

A. Computations

The processing of a task belonging to class C_k happens in two steps. First, it consumes messages received from tasks in the producing-class (C_j).

$$C_j = \begin{cases} C_N & \text{if } k = 1; \\ C_{k-1}, & \text{otherwise} \end{cases}, C_l = \begin{cases} C_1 & \text{if } k = N; \\ C_{k+1}, & \text{otherwise} \end{cases}$$

This step usually involves updating tables, generating a new population, filtering data elements or a combination of these. Second, it generates messages for tasks in the consuming-class (C_l). The task works as a mapper and performs a selection on the data (i.e., selection, fitting, crossover, classifying, categorizing and mapping, etc.). Therefore, the total number of computations performed by a task, $f_k(c)$, where $c \in C_k$, is a function of the processing done on messages received from tasks in C_j, and computation performed in generating messages for tasks in C_l, and is quantified in Equation 1.

$$f_k(c) = f(OD(c), ID(c)) \tag{1}$$

where $OD(c)$ is the number of messages sent by task c, and $ID(c)$ is the number of messages received by task c.

The function $f_k(c)$ is very application-specific and could be a combination of linear and non-linear terms (constant, polynomial, exponential, etc.). For example, in EpiSimdemics [6], the algorithm that models the spread of disease in population, the computation is performed by two classes of tasks. The number of computations performed by the task in the first class is linear to the number of generated messages, and the number of computations performed by the task in the second class is proportional to the number of received messages. In Section VI-A, we will quantify the number of computations performed by tasks in EpiSimdemics as a function of the number of incoming and outgoing messages, using application semantics and regression analysis. The procedure used is equally useful for all CPC algorithms.

B. Communication

The tasks exchange data in the form of messages. If the generating and receiving tasks are on the same processor, then the message is delivered locally. Otherwise, it is remote. As the overhead of the local message is negligible, we use the total number of remote messages exchanged across all processors to quantify the communication load. The communication may be overlapped with execution, and the overlap, if it happens, can make the cost estimation of the entire application difficult. The amount of communication-computation overlap is application-specific, and a general rule cannot be established to estimate it in advance. In Section VI-A, we use statistical regression modeling to capture the overlap in an example application (EpiSimdemics). The guidelines can be used to measure the communication-computation overlap in any CPC program.

IV. The Cost Metric

In this section, we develop a cost metric that quantitatively measures the cost of CPC algorithms. The cost metric is an extension of the min-max model developed in [15] and covers the algorithms/models where the computation happens in phases between classes of tasks. The key feature that enabled our approach is the communication load and synchronization separated computations. Figure 1 shows that due to synchronization, the execution cost of a simulation is dominated by the maximally-loaded or most-imbalanced processors (PE 0 by Class 1 tasks and PE 4 by Class 2 tasks). Therefore, we use the computational imbalances and communication load to estimate the cost of CPC parallel simulations as shown in Equation 2.

$$Mcost = k_1 IC_1 + k_2 IC_2 + ... + k_n IC_N + k_c CL \tag{2}$$

In the equation, IC_j is the computational imbalance in Class C_j tasks, and *CL* is the communication load. The constants, $k_1, \ldots, k_n$, and k_c show the relative contribution of components of the cost metric towards the cost of simulation. The metric is applicable to a broad class of CPC algorithms, where the nodes can be represented by a multi-partite interaction graph.

We quantify the computational and communication components of the cost metric in terms of incoming and outgoing messages (edges). Throughout the paper, we refer to the sum

of incoming and outgoing messages as the degree. We denote the total degree, in-degree and out-degree of a task c by $D(c)$, $ID(c)$, and $OD(c)$, respectively.

In distributed processing environment, the tasks are assigned to partitions where each partition contains one or more tasks of a particular class. Each task is assigned to exactly one partition. The computational load of partition i belonging to class C_k is denoted by *L(P_{ki})*. It is the sum of the computational loads of all the tasks in that partition and is quantified in Equation 3.

$$L(P_{ki}) = \sum_{c \in P_{ki}} f_k(c) \tag{3}$$

The processing of classes is separated in time. In an ideally load-balanced assignment of CPC tasks, each partition of a class will have an equal computational load (i.e., mean-load of class). The mean-load of C_k is denoted by $\overline{L_k}$ and is quantified in Equation 4.

$$\overline{L_k} = \frac{\sum_{i=1}^{R} L(P_{ki})}{R}, \tag{4}$$

where *R* is the total number of partitions. A partition is most imbalanced in terms of computations of class C_k if it has the maximum computational load. We measure the computational imbalance in C_k by IC_k, which is quantified in Equation 5.

$$IC_k = \frac{\max((L(P_{ki})) - \overline{L_k})}{tL_k} \tag{5}$$

$$tL_k = \sum_{i=1}^{R} L(P_{ki}) \tag{6}$$

In the Equations 5 and 6, tL_k is the summation of computational loads of all the tasks in C_k. The factor tL_k (the total Class-load) works as a normalization factor and keeps the imbalance in bounds. If IC_k is zero, then the computation of C_k is perfectly balanced across all the partitions, which means that every partition is assigned tasks (of C_k) with a total load of $\overline{L_k}$. In contrast, IC_k is equal to one, which means that the load of C_k is maximally imbalanced (, and this happens when one partition receives all its tasks).

After quantifying the computational imbalance, we discuss the communication load. Communication load is the ratio of remote messages to total messages. It is zero *CL* when there are no inter-process messages. In contrast, *CL* is one (maximum) when all the messages are remote.

Another important component of cost analysis is synchronization overhead, which we do not explicitly time, but it is covered by regression modeling that we perform in the next section.

V. Model Fitting and Extraction of Constants

We use the multi-variable regression analysis to fit the cost metric and determine its constants. The regression analysis is useful for complex algorithms where a simple analytical solution is hard to find. They generate concurrent activity per data element and unpredictable, fine-grain communication requests, which makes it harder to apply standard model reduction techniques. Here are the general guidelines for determining the cost metric and its constants for any CPC algorithm.

1. Identify cost metric components: The regression method expresses the response parameter (simulation time) as a linear combination of the predictor (independent) variables. Each application will have its own set of independent variables: one computational component for each class and one communication component. In the cost metric in Equation 2, $IC_1, \ldots, IC_N$ and *CL* are independent variables, while *Mcost* is the dependent variable.

2. Collect data for all variables: We need to collect data from running the target application on a number of data samples. The data samples are chosen to cover a wide range of input data and to help create a more generalized model. For regression analysis, the required number of samples are at least ten times more than the number of variables [16].

3. Check data for normality: Data collected in the previous step is checked for normality using histogram plotting in statistical softwares (i.e., R, SPSS and JMP, etc.). A well-shaped cosine plot shows that the data is normally distributed in ranges for all variables.

4. Perform model fitting: The model is extracted using cross-validation techniques [17]. This requires using two samples of data: *Sample1* for fitting the model and *Sample2* for validating the obtained-model. The data collected in step 1 is divided into two samples (about 50% each) by statistical software. Based on the analysis of correlation and scaler plots of *Sample1*, a multiple regression analysis is employed to fit the model. The model is determined using the residual plot and R-square value. In the residual plot, if the variances of errors are fairly evenly dispersed around the zero mean line across all the levels of predicted values, the assumption is not violated. The model can be improved by adding linear and non-linear terms (polynomial and exponential, etc.) of independent variables and applying transformations to them. The model fitting can also be performed automatically by statistical software.

5. Validate the model: The results of the predicted model determined using *Sample1* can be validated on *Sample2* by comparing the R-square values. R-square value shows the ability of independent variables in explaining the variability in the dependent variable. It varies between 0.0 - 1.0, where a value of 1.0 shows the perfect fit.

6. Check the model for over-fitting: It is very important to check the model for possible over-fitting. The over-fitting can be avoided in three steps. First, validate the model using cross-validation techniques [17]. Second, collect enough samples of the data (with dependent and independent variables) [16]. Finally, avoid over-interpretation by clearly stating the assumptions made (and providing enough details for another researcher to replicate the work).

In the next section, we apply the methodology discussed to determine the cost metric for EpiSimdemics.

VI. Example Algorithms

In this section, we apply regression modeling to determine the cost metric and its constants EpiSimdemics [6], [7], [14].

TABLE I
COEFFICIENSTS FOR COST METRIC COMPONENTS GENERATED USING STATISTICAL REGRESSION MODELING.

	Constants	Unstanderdized Coefficients	Std. Error	Standerdized Coefficients	T	Sig.	Collinearity Stats	
							Tolerance	VIF
$Constant$		.003	.000		27.332	.000		
IC_1	k_1	.051	.007	.061	6.946	.000	.994	1.134
IC_2_center	k_2	.747	.009	.833	81.855	.000	.652	1.534
$IC_2\_center\_sq$	k_{2b}	35.813	1.626	.181	22.023	.000	.994	1.006
CL	k_c	.004	.000	.201	19.467	.000	.631	1.585

We also perform the cost analysis of a combinatorial genetic algorithm, called ASPARAGOS [3].

A. EpiSimdemics

EpiSimdemics is a scalable parallel simulator that models the spread of contagion across a social contact network. The network is a bipartite graph containing person and location nodes. Each edge between a person node and a location node represents a visit to a location by that person. The persons produce messages and send them to locations to consume. The locations compute interactions (e.g., transmission of contagious disease) between all pairs of spatially and temporally co-located people and send the outcomes back to the persons. EpiSimdemics is a good example of CPC modeling, where person computation is performed by C_1 tasks and location computation is performed by C_2 tasks.

1) Load Analysis: Before applying regression analysis, the user needs to determine a quantification for the computational components in terms of incoming and outgoing messages. In EpiSimdemics, the number of messages that person tasks receive are negligible, we estimate the relative computation time of person tasks by the number of messages that they generate ($f_1(c) = OD(c)$). The number of messages that the location tasks generate are negligible, therefore the relative computation time of location tasks is estimated by the number of messages that they receive ($f_2(c) = ID(c)$). For space reasons, we omit the details of load analysis. We use imbalance definitions in Equation 5 and load definitions (of persons and locations) in this section to quantify IC_1 and IC_2 respectively.

Next, we apply statistical regression analysis to fit the model and determine constants for the three cost metric components of EpiSimdemics.

2) Data for Regression Analysis: Data for regression analysis is collected from running EpiSimdemics on 732 samples of three data sets, Alabama (AL), Florida (FL) and California (CA). Here is the methodology used to create data for regression analysis.

We first create 64 partitions of AL, FL, and CA sets using k-way and multi-constraint features of Metis [18], [19] partitioning scheme. Metis created partitions serve as base cases for generating other samples. The base cases give 18% remote communication, and close to balanced person and location loads (using definitions of person and location loads in Section VI-A1). We modify the base cases by randomly moving persons and locations from some of the tasks to other tasks to create a total of 732 samples. This creates different person imbalances (IC_1), location imbalances (IC_2) and communication loads (*CL*) in each sample. In the 732 samples, remote communication is varied over the range of 18% - 100% of total communication. IC_1 and IC_2 is varied over the range cover of 0% - 100% of mean person and location load respectively, thus forming a representative sample of the space of possible partitions. After collecting the data, we perform statistical modeling to determine the cost equation and its constants.

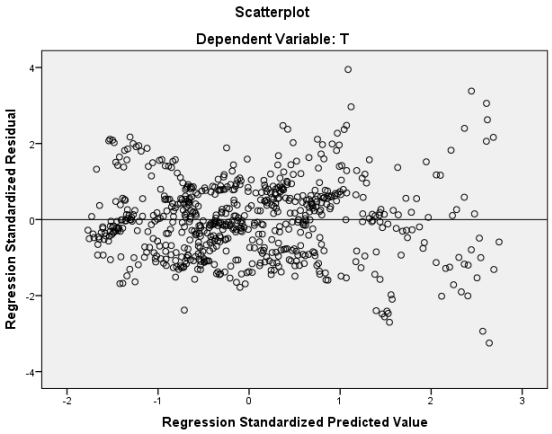

Fig. 4. Distribution of error do not critically deviate from the homoscedasticity for *Sample1*.

3) Model Fitting: The statistical analysis was conducted using SPSS [20] version 20.0 on 732 samples with four variables: simulation runtime (*T*), person computational imbalance (IC_1), location computational imbalance (IC_2) and communication load (*CL*). The histogram plot in Figure **??** illustrates that the data is fairly normally distributed for all variables. We divided the 732 samples randomly into two samples (approximately 50% by SPSS). According to the results of the preliminary analysis of *Sample1*, such as descriptive statistics and correlations between variables, a multiple regression analysis was applied to *Sample1* to fit a parsimonious model that best fits to the data.

As shown in Table I, the final model includes a quadratic term ($IC_2\_center\_sq$) for a mean-centered variable (IC_2-0.0083) as well as three linear terms (IC_2_center, *CL*, and IC_1). The quadratic term was determined while adding the square and cube terms of the independent variables to improve the fitness. The final model was reached when the variances of errors

were normally distributed around the zero mean line across all levels in the residual plot 4. Independent variables in the final model account for 97.8% of the variability in the dependent variable (r^2=0.956). In Table I, the regression coefficients of IC_2_center, $IC_2\_center\_sq$, IC_1, and *CL* are positive and statistically significant (*p-value*<0.001), implying that increase of each independent variable when controlling for other independent variables increases the simulation time (*T*). Analysis of normality shows that the data is fairly normally distributed for all variables.

4) Model Validation: We evaluate the results of the final predicted model on *Sample2*. To investigate how much variation in *T* in *Sample2* is explained by the regression model obtained from *Sample1*, we created a new variable, *predicted T*, using the equation from *Sample1* and then computed the correlation coefficient (r = 0.981). Comparing the R-square (0.962) from *Sample2* with the R-square (0.954) from *Sample1*, it is slightly larger unexpectedly rather than shrinking. This shows that our model is valid.

We avoided the possible over-fitting by following the three steps mentioned in Section V.

The equation is very application-specific and applicable to EpiSimdemics only, and can be used for all of its data-sets. The equation can be used for a varied number of processing units. But, it is machine-specific and EpiSimdemics will need a separate equation for each architecture. However, the methodology is general, and is applicable to all applications.

B. Genetic Algorithm: ASPARAGOS

ASPARAGOS [3] is an important parallel Genetic Algorithm, which uses poly-sexual voting to find optimum solutions to population genetics. The algorithm defines a genetic representation of the optimization problem and creates an initial population of individuals. The processing happens in two steps. In the first step, the individuals perform local hill climbing to increase their fitness. After the hill climbing, they send messages to all members of the neighborhood and the global best individual for mating. In the second step, the individual creates a new offspring (from mating) with genetic operator using crossover, from which an outcome message is sent to the visiting parent. If the offspring is fitter than the individual, the individual is replaced with it. The algorithm goes through iterations until the desired fitness is achieved by at least one individual. The numbers of individuals and the neighborhood size is fixed across iterations. However, the neighborhood for different individuals may overlap with each other.

This application is a good example of CPC modeling, where the hill climbing and selection is computed by C_1 tasks and crossover is computed by C_2 tasks. Each task is assigned a number of individuals, and each processor is assigned exactly one task of every class.

Class1 (C_1) Computation: Tasks in Class1 perform two computations: hill climbing and partner selection. Local hill climbing of individuals is done with a simple 2-opt exchange (placement of two processes is exchanged). The exchange happens if it improves the evaluation function. Hill climbing happens until no improvement happens. Its complexity is $O(n^2)$, where *n* is the size of neighborhood. In partner selection, the individual sends messages to all the individuals in the neighborhood and the global best individual. Therefore, we estimate the relative compute time of C_1 task by the number of its incoming and outgoing messages, called degree ($f_1(c) = D(c)$).

Class2 (C_2) Computation: Tasks in Class2 perform mating between individuals. The individual mates with all the members of the neighborhood and the global best individual. Since the neighborhood size is fixed, each task is performing similar amount of computation. Similar to C_1 tasks, the compute time of C_2 tasks can be estimated using their degree ($f_2(c) = D(c)$).

Communication Load: For mating, tasks in C_1 send messages to tasks in C_2. Similarly, tasks in C_2 receive outcome messages from tasks in C_1. Communication is local if the sending task and receiving task are on the same processor, otherwise it is remote. Communication load is measured in the number of remote messages exchanged in all processors. Since the neighborhood sized is fixed and the number of individuals are fixed, the communication load stays the same through iterations.

The cost metric for ASPARAGOS will have three components: two computational (one for each class) and one communication component. The constants are application-specific, and each genetic algorithm will have its own set of constants. The cost metric and its constants can be generated using regression modeling with data from different population sizes. The extraction process will go through model fitting, validation and over-fitting steps. More details about the procedure are listed in Section V.

VII. Experimental Evaluations

In this section, we show the ability of the cost metric in estimating simulation cost. Later in the section, we discuss the utility of the cost metric in optimal scheduling (in the assignment of the number of processors) when running multiple simulations in parallel.

A. Experimental Setup

Hardware: Our experiments were performed on a high-performance computing cluster consisting of 318 compute nodes. Each node has two octa-core Intel Sandy Bridge CPUs and 64 GB of memory. The cluster uses infiniband (40Gbps) technology for interconnections.

Data: We use EpiSimdemics to simulate the spread of H5N1 avian influenza across subsets of the social contact network of the US population. The algorithms presented were evaluated using North Carolina (*NC*) and California (*CA*) data sets. Each test is executed for 120 time-steps (i.e., simulated days).

B. Cost Estimation and Strong Scaling

In this section, we show the ability of the cost metric in estimating the execution times of EpiSimdemics. We present strong scaling numbers for *NC* and *CA* data. *NC* data can fit into the memory of a single node, while *CA* data requires a minimum of two nodes. Pre-partitioned data is assigned in a

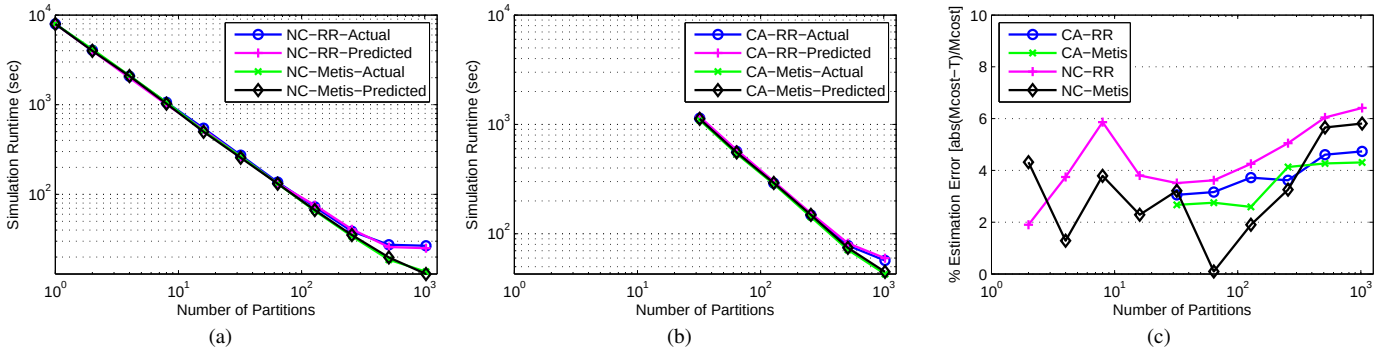

Fig. 5. Comparison of the cost metric predicted and actual execution times when simulating *NC* and *CA* data. The data was partitioned using Round-Robin and Metis schemes. (a) The cost metric estimates the running times of *NC* with very a small error. (b) The cost metric estimation is slightly better for the larger data, *CA*, than *NC*. (c) The maximum estimation error is 6.5% only (*NC* on 4K processors).

one-to-one mapping of partitions to processors. More details about the partitioning schemes (Round-Robin and Metis) and their performance in EpiSimdemics can be found in [19].

Figure 5 compares the cost metric estimated running times and actual running times for simulating *NC* and *CA* data. The matching slopes of the actual and predicted curves shows that the cost metric accurately estimates the strong scaling running times for most cases. The lines diverge differently for the two partitioning methods, showing the difference in performance. The cost metric was able to accurately predict this difference. The cost metric is slightly less accurate at predicting the execution times for a very large number of partitions. While a further study is required, one thought is that the synchronization overheads become dominant once a larger number of processors are in use [21], since the ratio of computation to communication decreases as we distribute data to more processors. *CA*, which is 4 times bigger than *NC*, shows a smaller error at high partition counts. Figure 5(c) shows the error in estimating the execution costs of *NC* and *CA*. The error increases up to 6.5% when estimating the execution time for *NC* on 4K processors. This shows that the cost metric predicts execution times even for a large number of processors with only a small error.

C. Resource Allocation using the Cost Metric

Since the cost metric predicts the strong scaling running times with small error, we can use it to perform the cost analysis of studies before launching the jobs. We can determine an optimal allocation of processors to run each data that minimizes the time to completion. Figure 6(c) shows that we get the best performance when both the jobs are executing in parallel: *NC* running on 896 processors and *CA* running on 3200 processors.

We showed the usefulness of the cost metric in resource allocation of *NC* and *CA*; however, the metric is general and can be used in job scheduling (resource allocation) of any data running EpiSimdemics. The job scheduling algorithms are well-studied, and we are not claiming any such contribution. But, the cost metric can be used as an estimation function to enable the job scheduler to determine optimal assignment. Moreover, it is not necessary that all the jobs need to run in parallel to minimize the time to completion. Rather, the jobs can run in groups (subset of jobs), and get the optimal

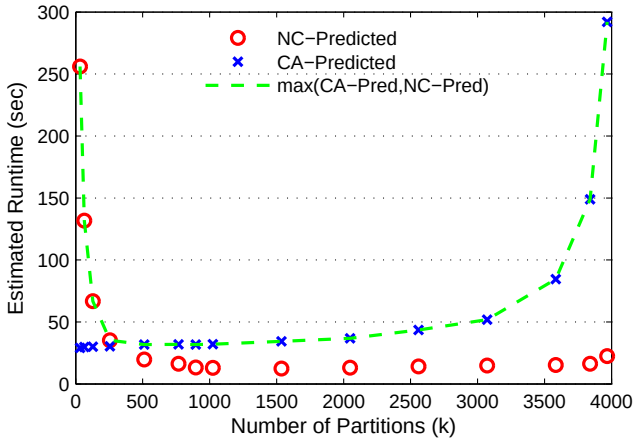

Fig. 6. Cost metric estimated execution times for running *NC* and *CA* data in parallel. *NC* was run on *k* cores and *CA* on *n-k* cores. *n*, the total number of cores (4K in this case), is fixed. The time to completion, max(CA-Pred,NC-Pred), is minimized the most when *NC* is run on 896 cores, and *CA* is run on 3200 cores.

performance (minimize the time to completion of running all the groups).

VIII. Related Work

Load analysis and cost estimation metric provide powerful tools for performing resource allocation, performance modeling and load balancing in constrained producer-consumer algorithms. Leslie [1], [2] introduced the bulk synchronous parallel (BSP) model for modeling of parallel algorithms. In BSP, the computation is performed in N step separated by synchronization. BSP is similar to CPC, but in CPC, the computational load may not be linear in terms of communication. Also, we perform a detailed cost analysis of CPC algorithms, which cover the computation-communication overlap, but the BSP model doesn't consider the computation-communication overlap. Shen et al. [22] proposed a cost function for evaluating the effectiveness of task assignment. The cost function measures the maximum time for a task completion. Although the method is useful for task-scheduling, it does not cover the cost analysis of CPC algorithms. Angelia and Asuman [23] presented an analysis of a distributed computation model for optimizing the sum of objective functions in multiple agents. The model

is good, but requires dynamic updates for cost estimation and is not practical for static cost analysis. Ding et al. [24] evaluated the min-max clustering principle for measuring the load balancing in partitions. Ercal et al. [25] also developed a cost function that the task assignment algorithms try to minimize. However, the models are not directly applicable to CPC algorithms that we consider in this paper. Our work provides a detailed statistical modeling for extraction of the cost metric and its constants.

IX. Discussions and Future Work

This paper presents a novel approach for quantifying the computational loads of CPC algorithms in terms of communication dependencies. Another important contribution of this work is the development of a cost metric for parallel CPC algorithms. The imbalance in workloads on different processors is known to be a bottleneck for high resource utilization. This is important in CPC programs, because the processing of classes is sequential, and the most loaded partition of every class makes the next class wait for it to reach synchronization. We use the most imbalanced partition of each class and remote communication to estimate the execution cost of CPC algorithms. The constants associated with components of the cost metric are very important and show the relative contribution of computational and communication loads in the total cost of simulation. Each application has its own set of constants, and we outline a detailed statistical methodology for extracting them. We illustrated the method for EpiSimdemics; however, the methodology is equally applicable to a broad class of CPC algorithms.

The cost metric predicts the strong scaling running times with small error and has wide application in resource allocation algorithms and performance evaluation of partitioning (load balancing) schemes.

Our short-term goal is to show the applicability of the cost metric to a variety of hardware, including peta-scale HPC systems. In long term, we want to extend our developed cost metric for use in dynamic load balancing of CPC simulations.

Acknowledgment

We thank members of the Network Dynamics and Simulation Science Laboratory (NDSSL) for their suggestions and comments. The authors acknowledge Advanced Research Computing at Virginia Tech for providing computational resources and technical support that have contributed to the results reported within this paper. This work has been partially supported by NSF PetaApps Grant OCI-0904844, NSF SDCI Grant OCI-1032677, NSF REU Supplement Grant CNS-0845700, and DTRA CNIMS Grant HDTRA1-07-C-0113.

References

[1] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[2] Alexandros V Gerbessiotis and Leslie G Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2):251–267, 1994.

[3] Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Parallelism, Learning, Evolution*, pages 398–406. Springer, 1991.

[4] John J Grefenstette, David E Moriarty, and Alan C Schultz. Evolutionary algorithms for reinforcement learning. *arXiv preprint arXiv:1106.0221*, 2011.

[5] Yasset Perez-Riverol, Roberto Vera, Yuliet Mazola, and Alexis Musacchio. A parallel systematic-monte carlo algorithm for exploring conformational space. *Current topics in medicinal chemistry*, 12(16):1790–1796, 2012.

[6] Christopher L. Barrett, Keith R. Bisset, Stephen G. Eubank, Xizhou Feng, and Madhav V. Marathe. Episimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[7] Keith Bisset, Xizhou Feng, Madah Marathe, and Shrirang Yardi. Modeling interaction between individuals, social networks and public policy to support public health epidemiology. In *Proceedings of the 2009 Winter Simulation Conference*, pages 2020–2031, December 2009.

[8] Kalyan S Perumalla and Sudip K Seal. Discrete event modeling and massively parallel execution of epidemic outbreak phenomena. *Simulation*, 88(7):768–783, 2012.

[9] Karthik Channakeshava, Keith Bisset, VS Anil Kumar, Madhav Marathe, and Shrirang Yardi. High performance scalable and expressive modeling environment to study mobile malware in large dynamic networks. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 770–781. IEEE, 2011.

[10] Keith Bisset, Md Alam, Josep Bassaganya-Riera, Adria Carbo, Stephen Eubank, Raquel Hontecillas, Stefan Hoops, Yongguo Mei, Katherine Wendelsdorf, Dawen Xie, et al. High-performance interaction-based simulation of gut immunopathologies with enteric immunity simulator (enisi). In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 48–59. IEEE, 2012.

[11] Charles M Macal and Michael J North. Agent-based modeling and simulation. In *Winter Simulation Conference*, pages 86–98. Winter Simulation Conference, 2009.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[14] K.R. Bisset, A.M. Aji, E. Bohm, L.V. Kale, T. Kamal, M.V. Marathe, and Jae-Seung Yeom. Simulating the spread of infectious disease over large realistic social networks using charm++. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 507 –518, May 2012.

[15] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Trans. Comput.*, 34(3):197–203, Mar 1985.

[16] Norman Richard Draper, Harry Smith, and Elizabeth Pownell. *Applied regression analysis*, volume 3. Wiley New York, 1966.

[17] Phillip I Good and James W Hardin. *Common errors in statistics (and how to avoid them)*. Wiley, 2012.

[18] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–13. IEEE Computer Society, 1998.

[19] Tariq Kamal, Keith R Bisset, Ali R Butt, Youngyun Chungbaek, and Madhav Marathe. Load balancing in large-scale epidemiological simulations. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 123–124. ACM, 2013.

[20] NY: IBM Corp. Armonk. Spss statistics for windows, 2011.

[21] Gene M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[22] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *Computers, IEEE Transactions on*, C-34(3):197 –203, march 1985.

[23] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009.

[24] C.H.Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and H.D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107 –114, 2001.

[25] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. *Journal of Parallel and Distributed Computing*, 10(1):35 – 44, 1990.