RC23588 (W0504-062) April 13, 2005
Computer Science

IBM Research Report

A Stochastic Newton Method for Control of Stream Processing and Multimedia Systems through Network-Level Actuation

Dimitrios Pendarakis, Jeremy Silber, Laura Wynter
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Stochastic Newton Method for Control of Stream Processing and Multimedia Systems through Network-level Actuation

Dimitrios Pendarakis, Jeremy Silber, and Laura Wynter

***Abstract*— We introduce a novel scheme for control of high performance distributed systems, seeking to achieve processing (CPU) goals through network controls. This is particularly valuable when the network becomes a constrained resource as happens in stream and multimedia processing systems. Our technique involves the development of a highly efficient and non-intrusive stochastic Newton-type algorithm for simultaneous approximation and optimization of the processing objective.**

I. Introduction

Large-scale distributed systems, such as stream processing and multimedia systems, are extremely data intensive. Successful operation of these systems critically depends on the ability to meet service level agreements on application performance. New, effective means for managing such large and heavily-loaded processing systems has become an urgent need.

While substantial prior work exists on controlling processing and networking resources separately, our work is novel in that it recognizes the dependencies between these resources. Rather than develop goals for network management, and goals for processing management and treat each through its own control mechanism, we propose a new, hybrid paradigm for achieving processing goals through network controls. This is particularly valuable when the network becomes a constrained resource.

In addition to the high load imposed on distributed systems, demands on the networking and processing resources vary widely across the applications at any point in time; some are compute-intensive but require little communication resources, while others are the opposite. Moreover, the dependency between application processing utilization and allocated bandwidth is not only complex, but subject to random perturbation. However, knowledge of this dependency is necessary to efficiently allocate networking resources. Hence, while interaction between multiple applications cannot be described analytically as a function of bandwidth, given a set of applications and their relative processor priorities, we propose to *learn* the implicit relation between processor utilization and allocated bandwidth.

IBM Watson Research Center, Hawthorne, NY, USA, {dimitris,jeremy.silber,lwynter}@us.ibm.com

The contributions of this work are as follows: (i) we introduce a very low-cost *dynamic learning* procedure to determine the relation between performance and the communication resources; (ii) we develop a novel stochastic Newton-type optimization algorithm that robustly and rapidly drives the system towards a desirable operating point through bandwidth control. Our algorithm scheme is designed to provide both *stability* and *adaptability* in the presence of incomplete information or noise.

With respect to relevant literature, the authors in [3], [4] consider the interaction of traffic with the network and propose using a simple feedback mechanism based on local buffer occupancy levels to control the progress of different processes. Multimedia applications have often been developed with build-in adaptation mechanisms to handle network or system congestion. [6] describes application-level quality adaptation techniques and [7] presents adaptive mechanisms for real-time applications. Abeni and Buttazzo in [8] propose a framework for dynamically allocating CPU resources to tasks whose execution times are not know apriori. The motivation for *learning* the tasks' CPU requirements is similar to our work; however the authors consider only the CPU bandwidth and not that of the network and do not address the dependency between the two resources.

This paper is organized as follows: in Section II we present the problem formulation and model. Section III summarizes the stochastic Newton-type algorithm that simultaneously *learns* the complex relation between processing power, bandwidth allocation and *optimizes* a target-level criterion and presents a convergence proof, and Section IV presents our experimental results.

II. Problem Formulation

Consider n applications running on a node, each corresponding to a process which uses (local) link bandwidth to send and receive data to and from one or more remote nodes. We make the following assumption.

Assumption 1: [Pseudo-Stationarity] The set of applications, $i = 1 \ldots n$ remains constant throughout a time epoch of given length, where the length exceeds the convergence period of the algorithm.

In other words, the behavior of the stochastic system has some minimal degree of stationarity, enough to allow our algorithm to learn the application resource requirements and drive the sytem to the target operating point.

While our method can be applied to any system management goal, we will focus on the goal of achieving a set of desired processing capacity (CPU) allocations among different applications. This goal is akin to what workload managers or load balancers try to achieve in distributed systems. Specificically, we assume a set of desired processing capacity (CPU) allocations, for n processes, $c_1, c_2, \ldots, c_n$, is provided; c_i denotes the percentage of processing resources allocated to process i. For an allocation to be feasible, $\sum_{i=1\ldots n} c_i \leq 1$; however, it is not necessary to *impose* this condition explicitly, as the actual CPU utilization levels are observed variables, rather than explicitly controlled; hence, the condition is always satisfied. We assume for the target CPU levels the following:

Assumption 2: The input target CPU levels, t_i, $i = 1 \ldots n$, satisfy $\sum_{i=1\ldots n} t_i \leq 1$.

Each of the n applications (processes) has an associated bandwidth allocation percentage, denoted by $b_1, b_2, \ldots, b_n$, where b_i denotes the percentage of (local) network bandwidth allocated to process i. Naturally, $\sum_{i=1\ldots n} b_i \leq 1$. In general, CPU utilization of any process, i is a complex function of the bandwidths, $b \in \Re^n$, allocated to *all* processes, $c_i : \Re_+^n \mapsto \Re_+$, which depends on overall system load, number of concurrent processes and their interactions, memory allocation, choice of network transport protocol, etc.

If the relation of an application's CPU usage to its allocated bandwidth were known, our optimization task would be relatively straightforward: we could then seek to allocate the bandwidth vector, b, that minimizes some norm of the CPU percentages and the target. In practice, however, the relation of CPU utilization to bandwidth is not a known and deterministic mapping.

The main goal and contribution of our work is the *joint learning and optimization* of a function of this mapping. We define the problem as one in which the CPU-bandwidth relation is initially some (simple) a priori approximation. Through our adaptive algorithm, the CPU-bandwidth relation is updated iteratively, thereby *learning* the shape of this surface as a function of the control variable, the bandwidth allocation vector, b.

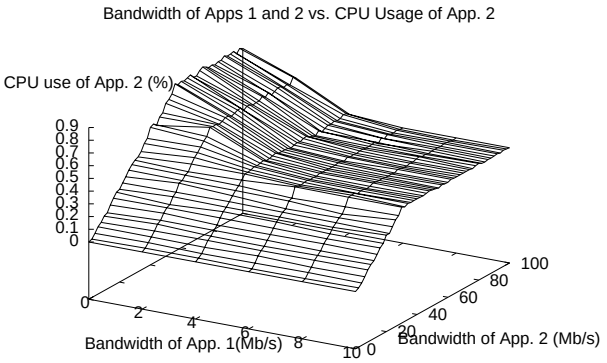

Fig. 1. Observed relation between bandwidth allocated to two different applications running concurrently and the CPU usage of one of the applications.

$$\frac{1}{2} E \left[\sum_{i=1\ldots n} [t_i - c_i(b)]^2 \right] = \frac{1}{2} E \|t - c(b)\|^2. \quad (1)$$

where, as before, the target CPU levels for each application i are t_i, $i = 1, \ldots n$.

Note that each step in the algorithm's operation will involve assigning a new bandwidth each application. This procedure causes perturbation and, as such, the number of iterations should be minimized. In addition, the CPU-bandwidth relation is not deterministic; rather it includes many sources of randomness including transient queueing effects associated and variations of an application's CPU utilization due to application state changes as well as random changes in (wide-area) network state.

Figure 1 illustrates the CPU utilization surface of one application as the bandwidth allocations for two applications running jointly on a node are increased. The piecewise-linear form of the curve is due to the sampling granularity in bandwidth space of the data. However, notice that the mapping C_1 is increasing individually in each variables, b_1 and b_2.

We shall optimize a distance criterion such as (1). We refer now to the *stochastic*, learned function of CPU usage, $C(b)$. The mathematical definition of the learnt CPU usage function will evolve at each iteration, j, of our algorithm. That is, for each $i = 1 \ldots n$,

$$C_i^j(b_i) = c_i(b_i) + \epsilon_i^j \quad (2)$$

where we assume that the ϵ_i^j are i.i.d. random variables, and where the *sequence of functions* $\{C_i^j(b_i)\}$ converges to $c_i(b_i)$ for all $i = 1, \ldots n$ as $j \to \infty$. Further, let $\bar{C}_i(b_i)$

be an empirical expectation of the CPU utilization for application i; that is,

$$\bar{C}_i(b_i) \sim c_i(b_i).$$

At each iteration, j, we shall seek to minimize

$$\begin{aligned} E[f^j(b)] &= \frac{1}{2}E\sum_{i=1\ldots n}(t_i - C_i^j(b_i))^2 \\ &= \frac{1}{2}E\|t - C^j(b)\|^2 \\ &= \frac{1}{2}\|t - \bar{C}^j(b)\|^2. \end{aligned} \quad (3)$$

The function (3) must be optimized subject to constraints on the control variable, b. That is, we impose the following set of polyhedral constraints:

$$b \in B := \{b \mid \sum_{i=1\ldots n} b_i \leq 1,\, b_i \geq 0, i = 1\ldots n\}. \quad (4)$$

The model defined by (3)–(4) is a stochastic, nonlinear program with polyhedral constraints. As we shall see in the next section, it is globally non-differentiable, due to the construction of the adaptive mapping, $\bar{C}$, but using our approach, the non-differentiability does not hinder the optimization procedure.

III. Adaptive Optimization and Approximation Algorithm

The objective of the algorithm is as follows: given a target CPU allocation vector, starting from an initial bandwidth allocation vector $b_1, b_2, \ldots b_n$, dynamically adjust the bandwidth allocations in such a way that the n applications consume the target amount of processing capacity. The solution we propose is thus to adaptively construct a learnt model of the $\{C(b)\}$ mapping. Over this evolving set of surfaces, our algorithm searches for the optimal operating point, given by the minimization of the Euclidean distance to the target operating point (3).

A. Adaptive model of the CPU utilization mapping, C

As shown in Figure 1, the CPU utilization surface exhibits some regularity, in spite of its complexity. In particular, we observe that $C_i(b)$ increases in b_i. The learning procedure is based on improving piecewise-linear approximations of each mapping, C_i^j, at each iteration, where j is the iteration number. Specifically, at iteration j, C_i^j is given by:

$$C_i^j(b_i) = \begin{cases} \alpha^{q(j-1)+1}b_i + \gamma^{q(j-1)+1},\ 0 \leq b_i \leq b_i^{q(j-1)+1} \\ \alpha^{q(j-1)+2}b_i + \gamma^{q(j-1)+2},\ b_i^{q(j-1)+1} \leq b_i \leq b_i^{q(j-1)+2} \\ \vdots \\ \alpha^{q(j)}b_i + \gamma^{q(j)},\ b_i^{q(j)} \leq b_i \leq 1, \end{cases}$$

where $q(j)$ is an iteration-dependent index, defined as follows:

$$q(j) = \sum_{i=1}^{j} i. \quad (5)$$

Hence, in iteration $j = 1$, the function $C_i^1(b_i)$ has only one slope, α_i^1. In iteration $j = 2$, there are two pieces, each potentially different from α_i^1, so that the pieces are numbered $1 + q(1)$ to $q(2)$ which, from (5), gives 2 to 3. Similarly, using (5), the pieces at iteration 3 are numbered 4 to 6, and so on.

For the system under consideration, the optimization problem (3) exhibits the following properties.

Proposition 1: The processing power function, $C_i^j(b_i)$ is piecewise linear, continuous, and nondifferentiable in b_i.

Proof: The mapping $C^j(b)$ is constructed iteratively. At the first iteration, $C_i^j(b_i)$ is a line passing through $(0, 0)$ and, possibly, the point $(1, 1)$, for each $i = 1, \ldots n$. In each iteration, when a new bandwidth point is determined, the resulting CPU utilization is measured and the slopes on either side of each C_i^j updated to insert the new point, i.e., the mapping, C_i^j will have j linear pieces at iteration j. Consequently, each C_i^j is continuous and piecewise-linear in b_i, its argument. Nondifferentiability follows from the piecewise-linearity.

We have described the CPU utilization function, C_i^j, in terms of a scalar argument, b_i, only. Although there are clearly cross effects, i.e., C_i indirectly depends on b_j, $j \neq i$, as mentioned previously, the algorithm treats the problem as if the functions were separable across applications, i and cross effects are dealt with as stochastic variations.

We shall make the following assumption for the sake of convergence of the algorithm, an assumption that has been empirically justified, e.g. in Figure 1.

Assumption 3: [Monotonicity of each $C_i^j(b_i)$] Assume that the mapping $C_i^j : \Re_+^n \mapsto \Re_+$ is increasing in its argument, b_i. for all $b_i \in B$, $i = 1 \ldots n$.

Typically, a CPU utilization C_i increases in b_i, but decreases in b_k, for $k = 1, \ldots n$, $k \neq i$. Cross derivatives are not used in our algorithm, though as CPU utilizations cannot exceed 1, they are implicitly present.

Remark 1: [Singularity of the Hessian of f] Due to the presence of the simplex constraint on the bandwidth vector, $\sum_{i=1,...n} b_i = 1$, the Hessian of the objective function is of rank $n-1$, since the nth term of the gradient, expressed as $f(1-\sum_{i=1,...n-1})$, is 0. To avoid singularity of the Hessian, it is sufficient to redefine the problem in a reduced space, of $n-1$ applications, the nth application bandwidth thus being derivable from the remaining $n-1$.

B. Steps of the stochastic Newton-type simultaneous approximation and optimization algorithm

Our approach is to replace the unknown CPU utilization function with a linear, separable approximation of it. Let $\bar{C}_i(b_i)$ be a smoothed estimate of the CPU utilization for application i; that is,

$$\bar{C}_i(b_i) = \frac{1}{m}\sum_{l=1}^{m} C_i(b_i, \epsilon), \tag{6}$$

for some number m of observations. We make use of multiple observations of the CPU utilization at a single bandwidth level to obtain an estimate of the expectation of the subgradient of C, ζ, with minimal system perturbation.

1) Initialization. Let the number of applications be n and the target CPU values be referred to by the $n-$vector, t. Set iteration counter, $j = 1$. Set initial bandwidth vector to a given starting point b_i^0 or set to $b_i^0 = 1/n$ for every $i = 1, ..n$ if no intial point is provided. Define the initial values $\bar{C}_i^0(LB(i)) = 0$ and $\bar{C}_i^0(UB(i)) = 1$, for all $i = 1, ..n$.
2) Main loop. While the stopping criterion has not been reached, repeat:
   a) Sample the CPU usage of each process i with the current bandwidth vector b. For each process $i = 1, ..n$, set $\bar{C}_i(b_i)$ to the smoothed CPU usage of process i.
   b) For each $i = 1, ..n$, if $\bar{C}_i^j(b_i^j)) > t_i$ then set UB(i)=j. Conversely, if $\bar{C}_i^j(b_i^j)) < t_i$ then set LB(i)=j.
   c) Direction finding. Determine the search direction, $g^j(b)$ such that $g^j(b) = (c^j(b^j) - t) * \zeta^j = (c^j(b^j) - t) * m^j$ where m is the vector of slopes of the piecewise-linear function $c(b)$, evaluated in the direction towards the target value, t; i.e., if, for application i, the current iteration counter j is the upper bound, then m_i^j is the gradient of the segment between point $\bar{C}_i^j(b^j)$ and $\bar{C}_i^{LB(i)}(b^{LB(i)})$ That is $m_i^j = \alpha_i^\ell$ for some active piece ℓ. Conversely, if j= LB(i), then m_i^j is the gradient of the segment between point $\bar{C}_i^j(b^j)$ and $\bar{C}_i^{UB(i)}(b^{UB(i)})$.
   d) Newton step. The Newton step is given by the (sub)-gradient scaled by the norm of the Hessian. Since we assume our objective to be seaparable, the norm of the Hessian is given, for each application i, by the second derivative of the objective function $f^j(b)$ evaluated at the active piece. hence, the Newton direction is given by $N_i(b^j) = (1/m^j)^2 g_i^j(b_i^j) = (c^j(b^j) - t)/m^j$
   e) Step size. Use a divergent-series step, $s^j = \gamma/(j+1)$, for some scalar constant, γ.
   f) Update. Set $b^{j+1} = b^j - g^j(b) * s^j$, and set $j = j + 1$.

A list of some or all $(b_i^j, \bar{C}_i(b_i^j))$ pairs may be kept in a sorted list for each $i = 1, ..n$. This makes it possible to quickly look up a bandwidth allocation b_i and estimated local slope m_i when a target t_i changes.

Figure 2 illustrates the steps of the algorithm. In the first step, the initial bandwidth value b_0 is set, and the CPU is measured as C_0. The upper bound (UB) is updated to be the current point, and a new bandwidth b_1 is chosen as the intersection of the target CPU level and the estimated function. In the second step, the CPU usage with bandwidth b_1 allocated is measured (C_1) and again is above the target, so the upper bound is set to the new point, and a new bandwidth b_2 is chosen. In the third step, we find that bandwidth allocation b_2 leads to a measured CPU usage C_2 below the target. The new point thus becomes the new lower bound (LB), and the next bandwidth allocation b_3 is chosen as the intersection of the target CPU line and the line between LB and UB. In the actual algorithm, this movement is moderated by a divergent-series step, which dampens the perturbations. The use of a Newton-type step as a search direction speeds up considerably convergence of the algorithm.

Proposition 2: Let

$$\zeta_i^j(b_i) = E[\xi_i^j], \tag{7}$$

where ξ_i^j is the ith component of a stochastic quasi-gradient of the CPU utilization mapping at iteration j. Then, $\zeta_i^j(b_i) = \bar{C}'(b)$ is a smoothed estimate of the stochastic quasi-gradient of $C^j(b)$ at iteration j.

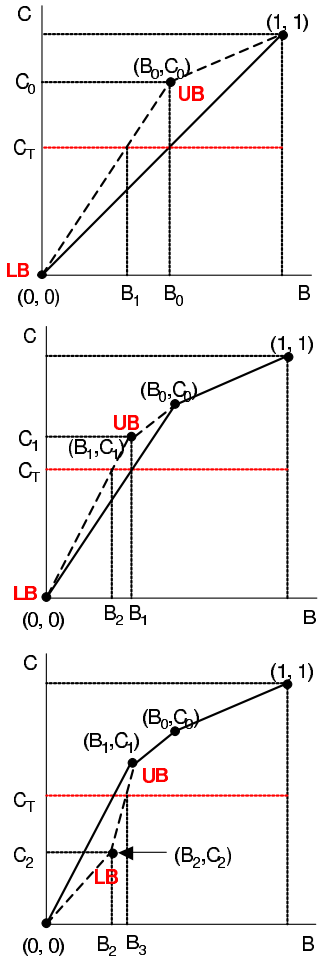

Fig. 2. Illustration of the steps of the algorithm.

Proof: Let the current iterate, b_i^j for application i and iteration j be such that $C(b_i^j) > t_i$. Then, an estimate of the expectation of a subgradient of C is, for some number m samples,

$$\begin{aligned}\zeta_i &= \frac{1}{m}\sum_{l=1}^{m}\xi_i^j(b_i,\epsilon), &(8)\\ &= \frac{1}{m}\left[\sum_{l=1}^{m}\frac{C_i^{l,j}-LB^j}{b_i^{l,j}-b^j(LB)}\right], &(9)\\ &= \frac{\bar{C}_i^j-LB^j}{b_i^j-b^j(LB)}, &(10)\\ &= (\bar{C}_i^j)'(b_i^j) &(11)\end{aligned}$$

where the second line comes from the definition of the subgradient of the piecewise-linear function C_i on the active segment, LB indicating the lower bound of the active segment. The last line follows from the fact that the samples are taken at the same bandwidth, hence $b_i^{l,j} = b_i^j$ for all $l = 1, \dots k$.

Under Assumption 3, we have the following property of the algorithm.

Proposition 3: If for any iteration, j, the stochastic quasi-gradient vector, $g^j = 0$, then the current iterate, b^j is a solution to optimization problem (3).

By construction, we have that, for each $i = 1 \dots n$, at every iteration, j,

$$C_i^j(LB_i^j) \le t_i \le C_i^j(UB_i^j). \quad (12)$$

In addition, due to the definition of the constraint set $\mathcal{B}$, we have that

$$LB_i \le b_i \le UB_i, \quad (13)$$

for every $i = 1 \dots n$. Under Assumption 3, the slopes of the piecewise CPU utilization mapping, $(C_i^j)'(b^j)^+ \ge 0$ and $(C_i^j)'(b^j)^- \ge 0$. Hence, if, at some iteration j, for every $i = 1 \dots n$,

$$\begin{aligned}g_i^j &= 0, &(14)\\ (t_i - C_i^j(b^j))m_i^j &= 0, &(15)\end{aligned}$$

then either $t_i - C_i^j(b^j)$, $i = 1 \dots n$, in which case the target has been reached, or $m_i^j = 0$, $i = 1 \dots n$. In the latter, there are two cases, (i.) $C_i^j(b_i^j) \le t_i \le C_i^j(UB_i^j)$ or (ii.) $C_i^j(LB_i^j) \le t_i \le C_i^j(b_j^i)$. In case (i.) if $m_i^j = 0$, then $C_i^j(UB_i^j) = C_i^j(b^j)$. Under Assumption 3, $C_i^j(UB_i^j) = t_i = C_i^j(b^j)$, and analogously in case 2, which completes the proof.

The following assumptions are needed to prove convergence of the algorithm.

Assumption 4: The set of stochastic quasi-gradients, at any iteration j, $\{g(b^j, \epsilon)\}$ is bounded. That is, for all j, ϵ, $\|g(b^j, \epsilon)\| \le \xi$, for some constant $\xi > 0$.

Assumption 4 holds when the feasible region, B is compact. See proposition B.24 in [1]. Hence, in our problem setting, Assumption 4 is always satisfied.

Assumption 5: [Functional convergence of approximation] The sequence of learnt functions, $f^j(b) \to f(b)$ uniformly over B.

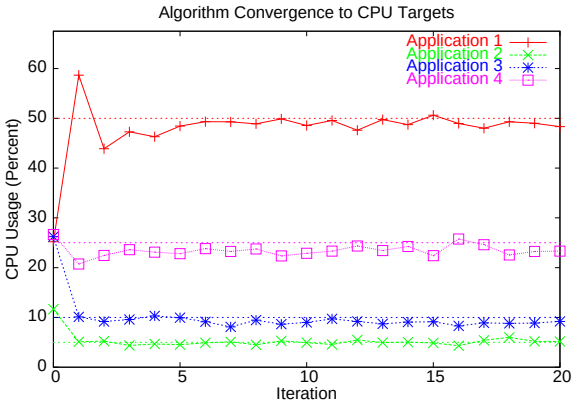

Fig. 3. CPU usage (top) and bandwidth usage (bottom) of four applications. Convergence in the control variable (Bandwidth) is very rapid. The CPU measure exhibits natural random variation as the application is running.

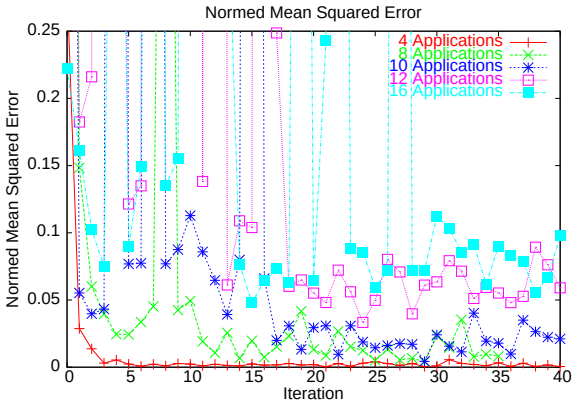

Fig. 4. Iteration vs Error (normalized sum of squared distances from target value) for controllers managing 4, 8, 10, 12, and 16 simultaneous processes.

Under Assumptions 4 and 5, we are able to prove the convergence of the algorithm to the set of optimal solutions of the original problem. In the convergence proof, we do not make use of the Newton-type search direction but rather a first-order subgradient direction, i.e., without scaling by the norm of the Hessian.

Theorem 1: [Convergence of the algorithm to optimal solution] Suppose that Assumptions 4 and 5 hold. By proposition 1, C_i^j are continuous for all $i = 1, \ldots n$. Suppose further that $f(b)$ and the learnt functions, $f^j(b)$, for all iterations j, are convex. Then, $f^j(b^j) \rightarrow f(b^*) = \min\{f(b) : b \in B\}$.

Proof: The feasible set, B, is convex and compact by contruction. The successive bandwidth vectors, b^j are given by the iteration $b^{j+1} = \Pi_B[b^j - s^j g^j(\zeta^j, b^j)]$, where $g^j(\zeta^j, b^j)$ is the expected subgradient of the objective function, $f^j(b^j)$, and the steps s^j satisfy $\sum_{j=1,\ldots\infty} s^j = \infty$, $\sum_{j=1,\ldots\infty}(s^j)^2 < \infty$. Hence, according to [2], the iterations of the simultaneous stochastic optimization and approximation procedure converge to the optimal solution value of the original problem.

IV. Experimental results

In order to produce a realistic testing environment, we experimented with a set of common stream processing applications that exhibit different CPU-bandwidth utilization functions, such as cryptographic, multimedia and text searching applications.

From Figure IV it is clear that, in the four-application instance, the control variable converges very rapidly. After some vaccilation around the target values, the CPU usage of each process settles in close to the target. CPU utilization for all four applications converges to within 5% of the targets in only three iterations, and stays within that range for the remainder of the experiment. Each iteration takes on the order of 2-5 seconds; hence overall convergence of the algorithm takes, in the case of 4 applications on the node, less than 15 seconds.

Figure IV shows the effect of increasing the number of processes under management on the convergence of the processes to their targets. Increasing the number of applications being controlled increases both the time required to reach convergence and the error measured when that convegence is achieved, although both criteria remain well within the range of acceptable for such a system. Indeed, convergence even for a relatively heavily-loaded node takes less than 1 minute. Typical nodes would run somewhere between these numbers of applications simultaneously, and convergence between 15 and 60 seconds allows for real-time use of this method in most instances.

In summary, the paradigm of using network filters as control variables in stream processing and multimedia systems was proven to be remarkably effective and accurate, in that we obtain convergence of CPU levels to within 5% of their targets, despite minimal fluctuations around those values and natural randomness in both measurements and CPU requirements of real applications. At the same time, the control/approximation algorithm which we have developed converged rapidly

enough to operate this level of control in a real-time stream processing or multimedia system.

REFERENCES

[1] Bertsekas, D. *Nonlinear Programming*, Athena Scientific, Belmont, MA, USA, 1998.

[2] Ermoliev, Y. "Stochastic quasi-gradient methods", Chapter 6 of *Numerical Techiniques for Stochastic Optimization*, Y. Ermoliev and R. J-B. Wets, Eds., Springer-Verlag, Berlin, 1988.

[3] Ashvin Goel, Molly H. Shor, Jonathan Walpole, David C. Steere, Calton Pu, "'Using Feedback Control for a Network and CPU Resource Management Application"', In Proceedings of the 2001 American Control Conference, Alexandria, Virginia, June 2001.

[4] David Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, Calton Pu, "'Control and modeling issues in computer operating systems: resource management for real-rate computer applications"', In Proceedings of 39th IEEE Conference on Decision and Control (CDC2000), Sydney, Australia, December 2000.

[5] Jiani Guo, Laxmi N. Bhuyan, "'Load Sharing in a Transcoding Cluster"', In Proceedings of IWDC, pp. 330-339, 2003.

[6] A. Fox, "'Adapting to Network and Client Variability via On-Demand Dynamic Distillation"', in Proceedings of Seventh Intl. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, MA, 1996.

[7] D. Rosu, K. Schwan, S. Yalamanchili, R. Jha, "'On adaptive resource allocation for complex real-time applications"', in Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 1997.

[8] Luca Abeni, Giorgio C. Buttazzo, "'Adaptive Bandwidth Reservation for Multimedia Computing"', in Proceedings of 6th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA), 1999.