Dynamic Control of Data Streaming and Processing in a Virtualized Environment

Junwei Cao, *Senior Member, IEEE*, Wen Zhang, *Member, IEEE*, and Wei Tan

***Abstract*—Performance of data streaming applications is co-determined by both networking and computing resources, and therefore they should be co-scheduled and co-allocated in an integrated and coordinated way. Dynamic control of resource scheduling and allocation is required, because unilateral redundancy in either networking or computing resources may result in the overprovision of it and the other may become a bottleneck. To avoid resource shortage as well as overprovision, in this paper, a virtualized platform is utilized to implement data streaming and processing. In this platform, fuzzy logic controllers are designed to allocate CPU resources; iterative bandwidth allocation is applied and is processing- and storage-aware to guarantee on-demand data provisioning. Experimental results show that our approach leads to higher application performance as well as higher resource utilization, compared with other resource scheduling and allocation methods.**

***Index Terms*—virtualization, data streaming, resource scheduling and allocation, fuzzy logic control.**

***Note to Practitioners*—Data streaming has become an important paradigm in many business and scientific applications, such as fraud detection in banking industry and gravitational-wave observation in astronomy. In this paper, we apply virtualization to provide better support for these applications, leveraging its advantage in resource on-demand allocation. A novel dynamic control method is proposed so that CPU and bandwidth can be co-scheduled and co-allocated since for data streaming applications these resources are tightly coupled from the performance perspective. In this control method, fuzzy logic control is applied for CPU allocation and an iterative algorithm is adopted for processing-, congestion- and storage-aware bandwidth allocation. We use fuzzy control because it does not rely on the mathematical modeling of an object and can implement human experts' heuristic knowledge via IF-THEN rules. Therefore it is desirable in our data streaming scenario, due to the variable coupling and heavily nonlinear nature of the system.**

Manuscript received July 10, 2011; revised Oct 16, 2011. This work is supported by National Science Foundation of China (grant No. 60803017) and Ministry of Science and Technology of China under National 973 Basic Research Program (grants No. 2011CB302505 and No. 2011CB302805) and National 863 high-tech Program (grant No. 2011AA040501).

Junwei Cao is with Research Institute of Information Technology and Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China (corresponding author, phone: 86-10-62772260; fax: 86-10-62795871; e-mail: jcao@tsinghua.edu.cn).

Wen Zhang was with National CIMS Engineering Research Center, Tsinghua University, Beijing 100084, China. He is now with Chongqing Military Delegate Bureau, General Armament Department of PLA, Chongqing 400060, China.

Wei Tan is with IBM T. J. Watson Research Center, Hawthorne, New York, 10532 USA (e-mail: wtan@us.ibm.com).

I. Introduction

Data streaming and processing has become more important in many business and scientific applications. Such applications require efficient transmission of data from/to distributed sources. It is often not feasible to store the entire data before subsequent processing because of the limited storage and high volumes of data to be processed. Most applications run continuously and require efficient use of computational resources to carry out processing in a timely manner [1]. An example is the LIGO (Laser Interferometer Gravitational-wave Observatory) [2] which aims at the detection of gravitational waves emitted from space sources. LIGO data analysis streams terabytes of data per day from observatories for real-time processing using scientific workflows [3]. This is a typical scenario in many scientific applications where data processing is continuously conducted over remote streams as if data were always available from local storage.

In our previous work [6], we have already demonstrated that, in data streaming applications, unilateral redundancy of either CPU or bandwidth does not necessarily lead to high throughput; on the other hand, shortage of either may also negatively impact the throughput. This reveals the tight coupling between CPU and bandwidth from the performance's perspective. Therefore it is required to allocate computing and networking resources in order to reach a balance between high throughput and high resource utilization. Virtualization technology which can dynamically provision virtual machines (VMs) [5], is a natural choice for resource management [4] in this scenario.

The contribution of this work is twofold. First a new co-scheduling framework is proposed on a virtualized platform, so that CPU and bandwidth can be co-allocated for LIGO streams processing. Such co-scheduling addresses the impact of the interplay between CPU and bandwidth on the system performance. Second, a fuzzy-logic based, closed-loop feedback control [7] method is devised for the aforementioned co-scheduling framework. Using this method we control VMs' CPU allocation by configuring VMs dynamically according to resource utilization, and iteratively allocate bandwidth by closely watching processing and storage status.

The rest of this paper is organized as follows: Section 2 formulates the data streaming and processing problem, and

Sections 3 and 4, fuzzy allocation of CPU resources and iterative bandwidth allocation are discussed, respectively. Experimental results are illustrated in Section 5 to show performance evaluation of our approach using a gravitational wave data analysis application. Section 6 discusses related work and Section 7 concludes the paper.

II. Data Streaming and Processing

A. Data Streaming Applications

Many existing scientific applications require data streaming and processing in a real time manner. Data sources can be large-scale simulators or observatories, with megabytes of data generated per second and terabytes of data aggregated per day. Data are usually streamed to remote processing nodes for various analyses. For these processing nodes, it is not feasible to store all data since new data constantly arrive and consumes local store space. Therefore, after data are processed and become obsolete, they need to be removed for newly arrival data. This typical scenario results in a tightly coupled relationship among *computing*, *storage* and *networking* resources.

For example, a LIGO gravitational wave data analysis application reads in two data streams from two remote LIGO observatories (one in Washington State and the other in Louisiana State) and calculates correlation coefficients that can be used to characterize similarity of two data curves. If two signals from two observatories occur simultaneously with similar curves, it would be likely that a gravitational wave candidate is detected. LIGO data are archived in specially formatted binary files, i.e., Gravitational Wave Framefiles (gwf). Each file is composed with 16s or 256s of data from multiple channels of an observatory. A LIGO data analysis application usually involves multiple data streams (series of small data files) from multiple observatories. LIGO data are archived in LIGO data grid [43] nodes which cannot provide enough computing resources for directly local data analysis. LIGO tries to benefit from open computing resources such as the Open Science Grid (OSG) [44]. While abundant computing resources are available in OSG, no LIGO data are available on OSG node. It becomes crucial to stream LIGO data from LIGO data grid nodes to OSG resources for large-scale processing. Data streaming and processing become essential for LIGO data analysis [38][39].

LIGO data analysis applications are developed using different operating and programming environments. In order to achieve finer-grained resource scheduling and higher utilization of computing resources, multiple applications have to further share one physical machine, and thus virtualization become the enabling technology. A virtualized environment can host multiple such data streaming applications. One single virtual machine (VM) provides a predictable and controllable run-time environment for each application. All computing, storage and networking resources on a virtualized platform can be shared among multiple VMs (and ultimately multiple applications), as illustrated in Fig. 1. While our previous work was focused on resource sharing among multiple physical machines [1], in this work, a virtualized environment is deployed, which brings new challenges on dynamic control.

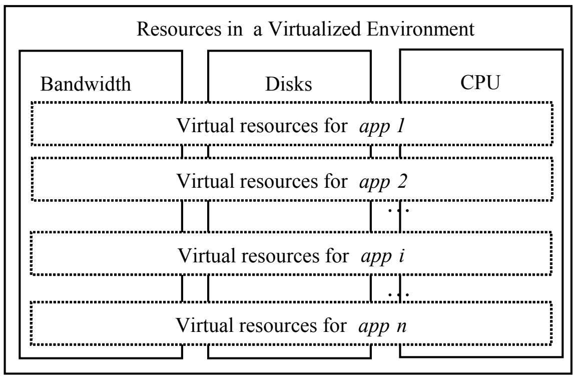

Resources in a Virtualized Environment
Bandwidth
Disks
CPU
Virtual resources for app 1
Virtual resources for app 2
Virtual resources for app i
Virtual resources for app n

Fig. 1. An illustration of resource sharing in a virtualized environment among multiple data streaming applications.

The local storage plays a key role correlating networking and computing resources. If no data is available, computing resources will be idle. If the allocated local storage is full for an application, data streaming cannot be carried out and networking resources cannot be utilized. At any time *t*, for a data streaming application *i*, if the amount of data in local storage, denoted as $Q_i(t)$, is higher than a certain level (e.g., a block as explained later), data processing is triggered. $Q_i(t)$ is co-determined by both data provisioning and processing since new data will be streamed to local storage while processed data will be cleaned up afterwards. The amount of output data (e.g. statistical values) is usually minor and ignored when we calculate local storage.

The amount of data in storage varies over time and can be described using the following differential equation:

$$\dot{Q}_i(t) = transpeed_i(t) - d_i(t) \qquad (1)$$
$$Q_i(0) = 0$$

, where $\dot{Q}_i(t)$, $transpeed_i(t)$ and $d_i(t)$ stand for the derivative of $Q_i(t)$, assigned transferring bandwidth and processing speed for data stream *i*.

If there are data available in the local storage, an indicator, denoted as $Ready_i$ for the application *i*, is set to be 1, otherwise $Ready_i$ is 0. So $d_i(t)$ can be described as:

$$d_i(t) = \begin{cases} 0, & Ready_i = 0 \\ > 0, & Ready_i = 1 \end{cases}$$

B. Performance Metrics

For data streaming applications, data throughput is the most important performance metric. Meanwhile resource utilization should be also considered.

Real Processing Speed (*RPS*): the actual data processing speed given by $d_i(t)$

Theoretic Processing Speed (*TPS*): the data processing speed the allocated CPU resources can generate if there were always sufficient data provisioned, denoted as

$procspeed_i(t,C_i(t))$, where $C_i(t)$ stands for the allocated CPU resource for application *i* at time *t*. Relationship between $procspeed_i(t,C_i(t))$ and $C_i(t)$ must be determined with system identification and it is obvious that $procspeed_i(t,C_i(t))$ is a non-decreasing function of $C_i(t)$, where $C_i(t)$ mainly refers to a proportion of CPU cycles as explained later.

Real Throughput (*RTP*): given a data provisioning scheme, the actual amount of data processed in a given period of time.

Theoretic Throughput (*TTP*): the amount of data processed in a given period of time if there were always enough data provisioning.

Scheduling CPU, storage and bandwidth resources is carried out periodically to deal with dynamic nature of resources and applications, and each period is referred to as a *scheduling period*. Suppose the length of a scheduling period is *M*, and for the h^{th} scheduling period, the following formulas are straightforward:

$$d_i(t)=d_i\left(t,C_i(t)\right) = \begin{cases} 0, & Ready_i=0 \\ procspeed_i\left(t,C_i(t)\right), & Ready_i=1 \end{cases}$$

$$TTP_{i,h}=\int_{(h-1)M}^{hM} procspeed_i\left(t,C_i(t)\right)dt$$

$$RTP_{i,h}=\int_{(h-1)M}^{hM} d_i\left(t,C_i(t)\right)dt$$

From (1):

$$RTP_{i,h} = \int_{(h-1)M}^{hM}\left(transpeed_i(t)-\dot{Q}_i(t)\right)dt$$

$$=\int_{t=(h-1)M}^{hM} transpeed_i(t)dt+Q_i\left((h-1)M\right)-Q_i(hM)$$

Define utilization of computing resource (*UC* in short) as

$$UC_{i,h}=\frac{RTP_{i,h}}{TTP_{i,h}} \qquad (2)$$

i.e.,

$$UC_{i,h} = \frac{\int_{t=(h-1)M}^{hM} transpeed_i(t)\,dt+Q_i\left((h-1)M\right)-Q_i(hM)}{\int_{t=(h-1)M}^{hM} procspeed_i(t)\,dt} \qquad (3)$$

, denoting to what extent the allocated compute resource is utilized.

$RTP_{i,h}$ can be defined in another form as:

$$RTP_{i,h} = \int_{\Omega i,h} procspeed_i(t)dt \qquad (4)$$

where $\Omega_{i,h}$ stands for the time fragment when processing is going on and then utilization can be redefined in another way as:

$$UC_{i,h} = \frac{\Omega_h}{M} \qquad (5)$$

Note that (5) implies that $TPS_{i,h}$ is a constant in a scheduling period with given CPU resources.

UC can be defined also as the ratio of *RPS* to *TPS*. The problem is to allocate proper amount of CPU resource to generate *RPS* approaching *TPS* as much as possible given the data supply scheme. It is obvious that redundant CPU resource will make a *TPS* much larger than *RPS*, which implies underutilization of computing resources. If available bandwidth is limited, *RPS* will be zero at most time with redundant CPU cycles for lack of data to process. This dependency between data provision and processing make it necessary to allocate compute resources on demand so as to make *RPS* as close to *TPS* as possible.

III. CPU Allocation with Fuzzy Control

CPU allocation is implemented using a fuzzy control approach on top of virtualization technology, where the virtualization provides an isolated run-time environment and the fuzzy control addresses appropriate resource configuration in a virtualized environment.

A. Virtualization with Xen

Recent progress on virtualization technology makes it possible for resource isolation and performance guarantee for each data streaming application. Virtualization provides a layer of abstraction in distributed computing environment, and separates physical hardware with operating system, so as to improve resource utilization and flexibility.

Xen [36], an open source hypervisor, is use to build the virtualized environment. With Xen, configuration of VMs can be dynamically adjusted to optimize performance. The CPU of a VM is called virtual CPU, often abbreviated as VCPU. The quota of physical CPU cycle a VCPU will get is determined by two parameters, i.e., *cap* and *weight*. The cap value defines the maximum percentage of the CPU that can be used by the VM; when multiple VMs compete for one CPU, their weight values define their proportion in getting the shared CPU. For example, a VCPU with a weight of 128 can obtain twice as many CPU cycles as one whose weight is 64, while 50 as a cap value indicates that the VCPU will obtain 50% of a physical CPU's cycles. In this work, *cap* is adjusted dynamically according to the measured utilization and pre-defined fuzzy rules as described below.

B. Fuzzy Control

A fuzzy control system [37] is based on fuzzy logic related with fuzzy concepts that cannot be expressed as true or false but rather as partially true. A fuzzy logic controller (FLC) is depicted in Fig. 2; it consists of an input stage, a processing stage, and an output stage.

Some basic concepts are given below to help construct an elementary understanding of fuzzy logic controllers and their mechanism.

Universe of discourse is the domain of an input (output) to (from) the FLC. Inputs and outputs must be mapped to the universe of discourse by quantization factors (*Ke* and *Kec* in Fig. 2) and scaling factor (*Ku* in Fig. 2), respectively, which helps to migrate the fuzzy control logic to different problems without any modification.

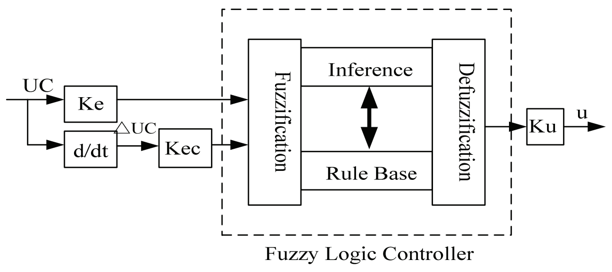


Fig. 2. A fuzzy logic controller. The input variables in a fuzzy control system are in general mapped into fuzzy sets, where an input variable may be mapped into several fuzzy sets with corresponding truth values determined by the membership functions. This process is called *fuzzification*. All the rules that apply are invoked, using the membership functions and truth values obtained from the inputs, to determine the result of the rule. This result in turn will be mapped into a membership function and truth value controlling the output variable. These results are combined to give a specific answer by a procedure known as *defuzzification*.

Linguistic variables describe the inputs and output(s) of a fuzzy controller. These linguistic variables are a natural way resembling human thoughts to handle uncertainties. Linguistic variables involved in this work include the input of the fuzzy controller, *UC* and*△UC* and the output of the fuzzy controller, a proportional factor (*PF)*.

Linguistic values are used to describe characteristics of the linguistic variables. Very low, low, medium, high and very high are the linguistic values for *UC*, while those for *△UC* and *PF* are *NB*, *NM*, *NS*, *ZE*, *PS*, *PM* and *PB*, where *N*, *P*, *B*, *M*, *S* and *ZE* are abbreviations of *negative*, *positive*, *big*, *medium*, *small* and *zero*, respectively, and the combination of them just takes on a degree of truth. Different from classical mathematics, in fuzzy world, this is represented as a continuous value between 0 to 1, and 0.5 indicates we are halfway certain. The mapping from a numeric value to a degree of truth for a linguistic value is done by the membership function.

Linguistic rules form a set of IF premise THEN consequent rules to map the inputs to output(s) of a fuzzy controller, i.e., to guide the fuzzy controller's actions. These rules are defined in terms of linguistic variables, different from the numerical input-or-output of the classical controller. A linguistic rule for example is: IF *UC* is *high* AND *△UC* is *NB* THEN *PF* is *NB.*

Rule-base holds a set of IF-THEN rules as a part of the controller, dictating how to achieve *PF* according to the fuzzified linguistic values of *UC* and *△UC.*

Membership functions quantify the certainty an *UC* and *△UC* value to be associated with a certain linguistic value. Except for the membership of linguistic value *very low* for *UC*, we use symmetric triangles of an equal base and 50% overlap with adjacent *MFs*. Unlike traditional set theory, in fuzzy set theory underlying fuzzy control theory, set membership is not binary but continuous to deal with uncertainties. Thus, a fuzzy input or output may belong to more than one set—maximum two adjacent sets in our MFs—with different certainty values.

Inference mechanisms in Fig. 2 determine which rules will be applied at the k^{th} sampling point, based on the fuzzified *UC* and *△UC.* To compute the certainty value of the premise in the corresponding IF premise THEN consequent rule(s), we take the minimum between the certainty values of *UC* and *△UC*, since the consequent cannot be more certain than the premise.

Fuzzification is to transform precise values of inputs into fuzzy sets with corresponding membership functions, which is indispensable for fuzzy inference. Outputs of fuzzy inference must be transformed into a clear value by *defuzzification.*

C. Linguistic Variables and Fuzzy Rules

As for the FLC proposed in this paper, the inputs are the observed resource utilization *UC* as defined in (2) or (5) and the derivative of it, *△UC*. Although it has two inputs, essentially it is a single input controller since *△UC* can be derived from *UC*, *△UC=UC(k)-UC(k-1).*

The output of the fuzzy controller for application *i* in the h^{th} scheduling period is a proportional factor, denoted as $PF_{i,h}$. At the h^{th} scheduling period, given inputs *UC* and *△UC*, suppose the relevant fuzzy sets of output form a set denoted as $M_{i,h}$ with membership denoted as $M_{i,h}(u)$, where $u \in U_{i,h}$ and $U_{i,h}$ is the universe of discourse, then the output can be calculated as

$$PF_{i,h} = \int_{U_{i,h}} M_{i,h}(u)u\,du \Big/ \int_{U_{i,h}} M_{i,h}(u)\,du \qquad (6)$$

Suppose the initial caps of each application are $C_{i,0}$, in the h^{th} scheduling period, the cap will be

$$C_{i,h} = C_{i,h-1} + \left(PF_{i,h-1} - 1\right) CapScale,\ h \ge 1 \qquad (7)$$

where *CapScale* is the varying scale of the allocated cap. $PF_{i,h}$ is adjusted every scheduling period, so it is adaptive to the varying situations. Relationship between the allocated cap and *procspeed* is a linear model, as described later in Section V.

$$procspeed(h) = \sum_{l=1}^{p} a_l\, procspeed(h-l) + \sum_{m=1}^{q} b_m Cap(h-m) \qquad (8)$$

Triangular membership functions are adopted for inputs *UC* and *△UC* and output *PF*, as shown in Fig. 3.

Table I provides a summary of fuzzy rules. A low utilization implies that the allocated CPU quota should be decreased to release redundant compute resources without reducing the ultimate throughput; meanwhile extremely high utilization indicates that more CPU resources are required to increase processing efficiency. When the utilization is *very low*, *low* or *medium*, the generated *PF* should be less than 1 while the very high utilization requires a *PF* lager than 1. To avoid oscillation, *△UC* should also be paid attention. When the utilization is far away from the settled point (80% here), the adjustment can be big. Then the *PF* in the 1st, 2nd and 3rd columns in Table I will be *NB* (negatively big) while in the last column it is *PB* (positively big). When the utilization falls to the high area, more careful adjustment is required, as shown in the 4th column in Table I. For example, when *△UC* is *NS*, *PF* is also *NS*; and when *△UC* is *ZE*, *PF* is also *ZE*, which means that no adjustment is required so as to keep a stable status.

From Table 1, it can be seen that these fuzzy rules are simple but robust. It can guarantee a rapid convergence to the settled point without stable state errors, which is a required characteristic for control systems as shown in experimental results included in Section V.

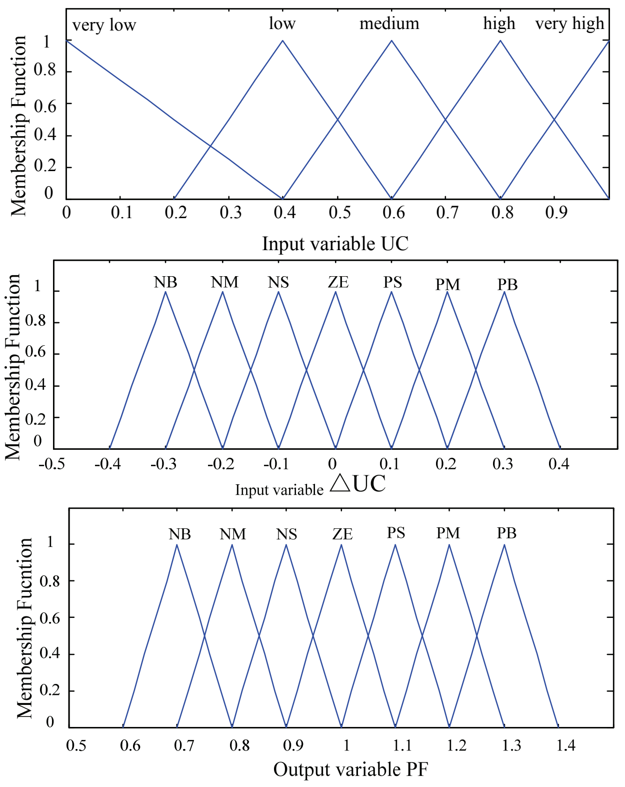


Fig. 3. Triangular membership functions of inputs *UC* and *△UC* and output *PF*. Linguistic values of *UC* include *very-low*, *low*, *medium*, *high* and *very-high*, indicating CPU utilization. Both input *△UC* and output *PF* adopt triangular membership functions with linguistic variables of *NB*, *NM*, *NS*, *ZE*, *PS*, *PM* and *PB*. The universe of discourse of *△UC* falls to the scope of -0.4 to 0.4, which is based on our empirical observation. It is also the case for *PF* where the universe of scope is set to 0.6 to 1.4.

TABLE I. FUZZY RULES

PF		***UC***				
		very low	*low*	*medium*	*high*	***very high***
	NB				*NB*	
	NM				*NM*	
	NS				*NS*	
△ UC	*ZE*	*NB*	*NB*	*NB*	*ZE*	***PB***
	PS				*PS*	
	PM				*PM*	
	PB				***PB***	

D. Resource Co-scheduling and Co-allocation

Dynamic control of resource co-scheduling and co-allocation for data streaming applications is illustrated in Fig. 4. CPU and bandwidth resources are co-allocated by the actuator *ACT*. The transfer function *G* from allocated CPU and bandwidth resources to utilization *UC* is not available, for the two inputs are tightly coupled with each other. Fortunately, this transfer function is not indispensable for our fuzzy allocation scheme.

A FLC receives *UC* and *△UC* and outputs the caps of CPU for each application and then *procspeed* is determined using (8). An iterative bandwidth allocation (IBA) is implemented as described in Section IV to decide *transpeed*. *UC* at the next scheduling period is obtained with (2) or (5). In such a way, the control system works and dynamic resource allocation is implemented for virtualization.

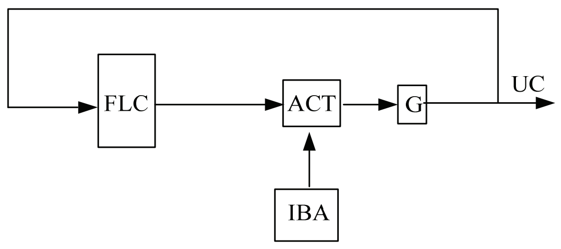


Fig. 4. Dynamic control of CPU and bandwidth resource co-scheduling and co-allocation.

IV. ITERATIVE BANDWIDTH ALLOCATION

Since data are streamed to local storage through network, multiple data streams need to share the total bandwidth of the virtualized environment, denoted as *I*. The individual data streams, called sessions, denoted as *s*, form a set *S*. Each session will be assigned with a bandwidth x_s (i.e., *transpeed* in Section II), where $x_s \in X_s$, $X_s = [b_s, B_s]$ and $b_s > 0$, $B_s < \infty$. b_s stands for the least bandwidth required for session *s*, while B_s is the highest bandwidth of the connection from the corresponding data source to session *s*. Session *s* has a utility function $U_s(x_s)$ that is assumed to be concave, continuous, bounded and increasing in the interval $[b_s, B_s]$. We try to maximize the sum of the utilities of all the sessions, maintaining fairness among them. The problem can be described as follows.

$$\max \sum_{s\in S} U_s(x_s)$$

s.t. $$\sum_{s\in S} x_s \le I \qquad x_s \in X_s$$

Variables U_s and L_s are the pre-defined upper and lower bounds of data volume in storage for session *s* to control the pause and resuming of data transfers, i.e., when the data volume in storage of *s* reaches the upper bound, data transfer is halted and when this volume reaches the lower bound, transfer is resumed. By this means data transfer may be intermittent rather than continuous, so as to be *storage aware*, to avoid data overflow while guaranteeing data provisioning.

According to the amount of data in storage, there are two possible transfer states for each *s* at any time, i.e., active and inactive. All active sessions form a set, called S_A, and it is obvious that this set is varying because transfer states of sessions are changing. At every sampling time *k*, status of transfer *s* can be determined by data amount in storage and the previous status:

$$status_{s,k} = \begin{cases} 1, amount_{s,k-1} < U_s, status_{s,k-1} = 1 \\ 0, \quad amount_{s,k-1} \geq U_s \\ 0, amount_{s,k-1} > L_s, status_{s,k-1} = 0 \\ 1, \quad amount_{s,k-1} \leq L_s \end{cases}$$

, where $status_{s,k}$ and $amount_{s,k}$ stand for status of transfer and data amount in storage for session s at the k^{th} sampling time, respectively. The initial condition is $status_{s,0}$ =1 and $amount_{s,0}$=0. We only allocate bandwidth for active transmissions, so the bandwidth constraint becomes:

$$\sum_{s \in S_A} x_s \leq I \cdot$$

An iterative optimization algorithm is proposed as follows.

While $s \in S_A$

$$x_s^{(k+1)} = \begin{cases} \left[x_s^{(k)} + \alpha_k U'\left(x_s^{(k)}\right) \right]_{X_s} \ if \ \sum_{s \in S_A} x_s^{(k)} \leq \rho I \\ \left[\beta_k x_s^{(k)} \right]_{X_s} \ if \ \sum_{s \in S_A} x_s^{(k)} > \rho I \end{cases}$$

otherwise,

$$x_s^{(k+1)} = 0, \forall s \notin S_A$$

Here, $x_s^{(k)}$is the bandwidth for session $s \in S$ at the k^{th} sampling time. $\{\alpha_k\}$ and $\{\beta_k\}$ are two positive sequences, $\beta_k \in$ (0, 1). $[\cdot]_{Xs}$ denotes a projection on the set X_s:

$$[y]_{X_s} = \min(B_s, \max(b_s, y))$$

$U'(\cdot)$ is the sub-gradient of

$$U(\cdot) = \sum_{s \in S_A} U_s\left(x_s^{(k)}\right) \text{ and } U'\left(x_s^{(k)}\right) = \frac{\partial U(\cdot)}{\partial x_s^{(k)}}$$

And ρ is the so-called safety coefficient to avoid bandwidth excess, where $\rho \in$ (0, 1).

This allocation algorithm is also *processing-aware*, since $status_{s,k}$ and $amount_{s,k}$ are affected by processing, i.e., S_A is associated with data processing. On the other hand, the allocated bandwidth will also change UC to trigger the FLC to adjust allocation of CPU resource. In this way, bandwidth and CPU resources are co-scheduled and co-allocated.

Parameters in this iterative bandwidth allocation scenario, such as α_k, β_k and ρ can be adjusted according to different allocation principles, such as relative fairness and the-most-needed-the-first. The most applied utility function is logarithmic:

$$U_s(x_s) = w_s \ln(1 + x_s)$$

, where w_s is the weight of session x_s and a larger w_s implies a bigger quota in the total available bandwidth.

Due to the coupling of data provisioning and processing, the relationship between processing efficiency and allocated CPU or bandwidth is heavily non-linear and is hard, if not impossible, to be expressed with a closed-form formula. It is difficult to apply traditional feedback control in absence of a precise model, and that is why in Section III we proposed to use fuzzy control as an alternative, due to its model-free nature.

V. Performance Evaluation

A. *Experiment Configuration*

VMs are set up on a HP DL580G5 server with 4 Intel CPUs containing 16 Xeon E7310 cores and 8 GB memories for LIGO data streaming applications. Data items are streamed to the VMs from remote data sources. LIGO data streams are composed with numerous small data files, each containing observational data acquired in 16 seconds. Here 1,188 pairs of LIGO data files from two observatories with the total amount of 4,354 MB are used in the experiment as sample data. In the following experiments, these data are utilized periodically when needed.

To reveal the mathematical relationship between the processing speed and the allocated computing resources (mainly the quota of CPU), system identification is carried out. The makespans during which all the data pairs are processed are shown in Fig. 5, from which it seems that allocated memory has little impact on the makespan for this application.

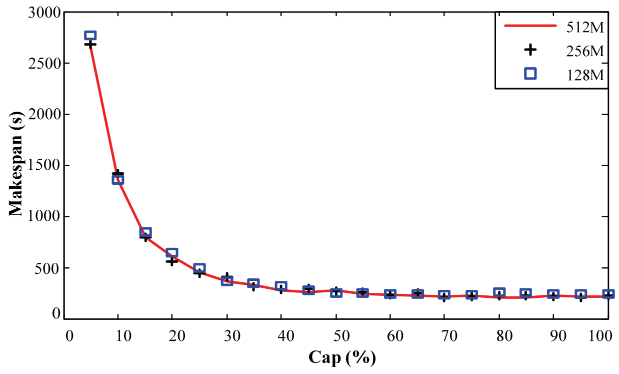


Fig. 5. Makespans with different caps and memory sizes. Three VMs are setup with 512 MB, 256 MB and 128 MB of memory, respectively. The allocated caps for each VM range from 5% to 100%.

The processing speed is shown in Fig. 6 with the solid line. Polynomial curve fitting is applied to generate a mathematical function from the cap to processing speed, using the Least Squares Method (LSM).

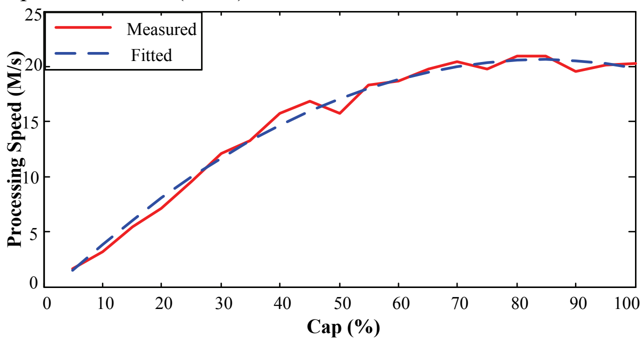


Fig. 6. Processing speeds with different caps. Memory size for each VM is set to 128 MB.

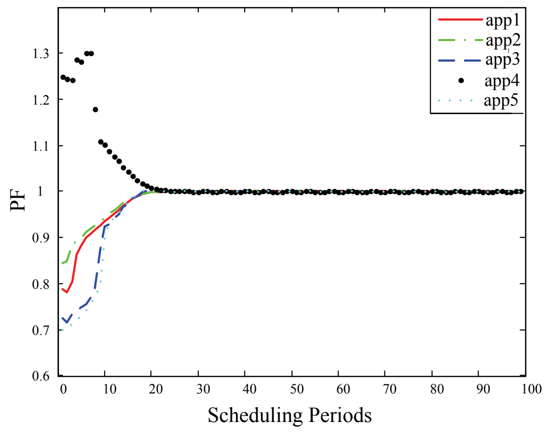
Figs. 5 and 6 infer that once the allocated cap exceeds 50%, the same increase in CPU Cap will not gain so obvious benefit as it does in the range of 5%-50%.

In our experiments, applications can be divided into two categories: *light* and *heavy* ones. In light applications, data items are processed with small amount of computation while heavy applications perform more complex data processing and relatively larger amount of computational resources are required. Heavy and light groups of applications are used as benchmarks, each group with 5 applications. Total available bandwidth *I*, set to 5, 10 and 15 Mbps, is shared among all applications. Three values of *CapScale*, 3, 5 and 8, are evaluated. Our allocation algorithm is evaluated for 100 scheduling periods. Each period lasts for 100 seconds in these experiments.

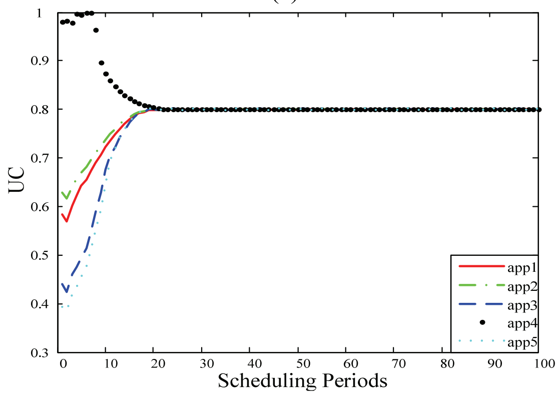
Performance metrics include the output of the FLC (*PF*), the utilization of each allocated computational resources (as define in (1), *UC*), the allocated cap for each application and resource usage (*US*) of CPU and bandwidth. Note that the usage of CPU means the sum of allocated caps for each application, which is different from the utilization defined in (1).

B. *Resource Allocation and Utilization*

Figs. 7 and 8 illustrates the proportional factor (PF), allocated resources (UC), cap value (Cap) and resource usage (US) values of heavy and light applications, respectively. As shown in Figs. 7 and 8, light and heavy applications get appropriate amount of CPU resources using our approach, where the total bandwidth is 5 Mbps. In these cases, required CPU resources of each application is far less than a whole physical CPU, because the allocated caps of them are under 20% or even 10%. The total CPU utilization is also far less than 100%.

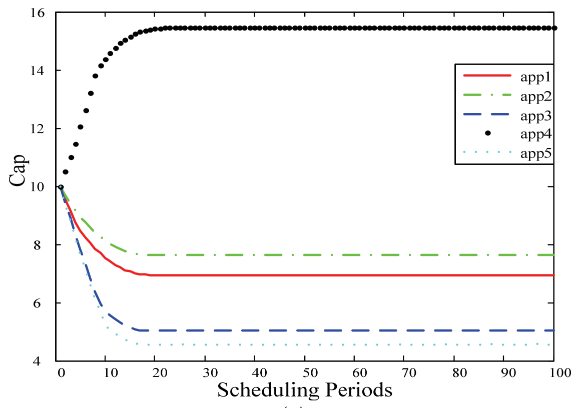


(a)

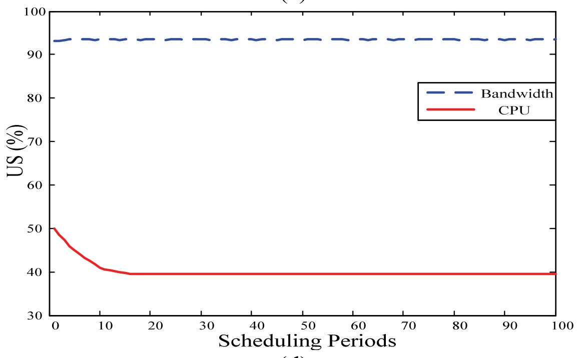


(b)

Fig. 7. (to be continued)



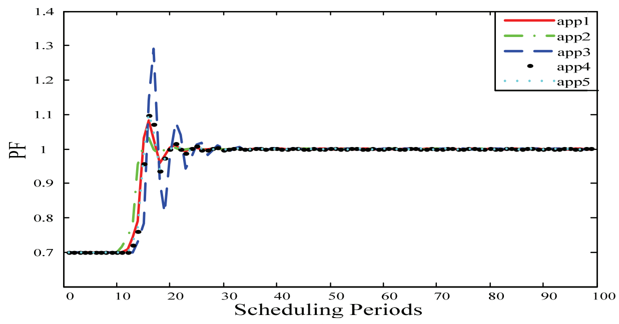
(c)



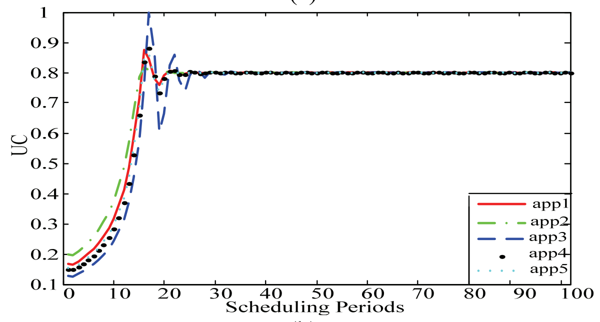
(d)

Fig. 7. Performance of heavy tasks. (a) Proportional factors; (b) Utilization; (c) Allocated caps; (d) Resource usage.

Initial caps for each application are 10%. All the allocation schemes are converged to a steady state. No steady state error exists in each allocation scheme. Also in presence of sudden changes of available resources, they can make a rapid response, as shown in Fig. 8.

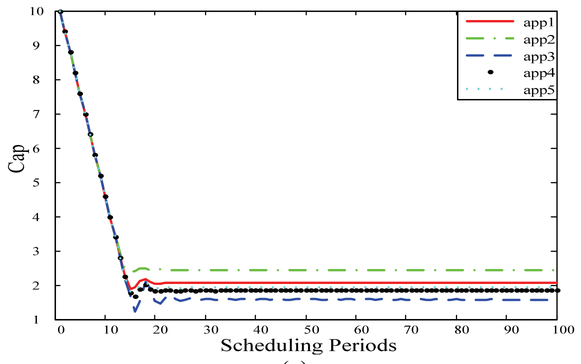


(a)

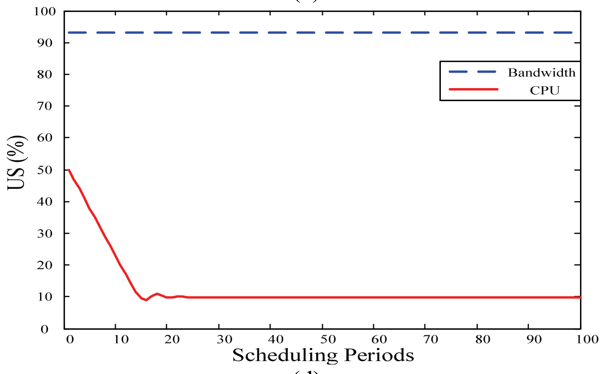


(b)

Fig. 8. (to be continued)



(c)

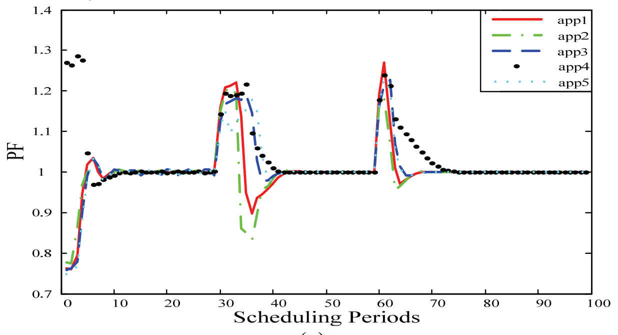


(d)

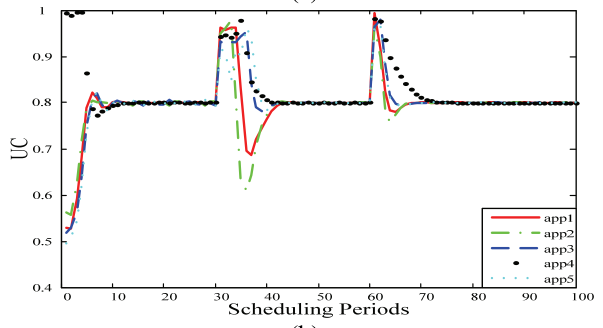
Fig. 8. Performance of light tasks. (a) proportional factors; (b) utilization; (c) allocated caps; (d) resource usage.

C. *Robustness and Adaptability*

Adaptability of the allocation algorithm is also tested. Total available bandwidth jumps to 8 and 10 at the 30th and 60th scheduling period, respectively. Performance metrics are used the same as those in Figs 7-8, but here we only show results with heavy tasks.

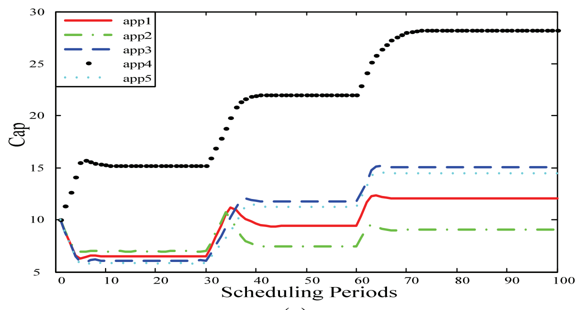


(a)

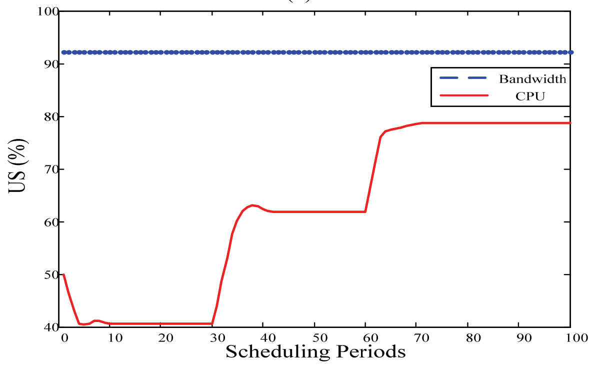


(b)

Fig. 9. (to be continued)



(c)



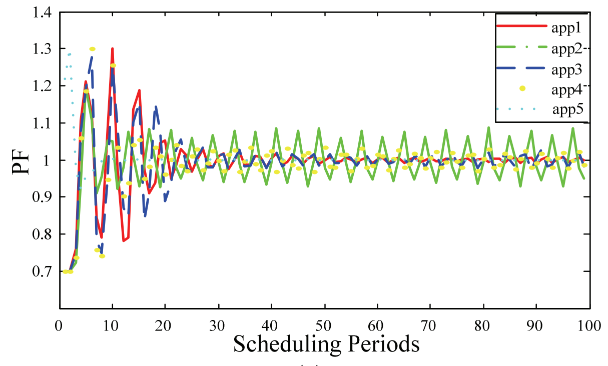
(d)

Fig. 9. Responsiveness to bandwidth changes with heavy tasks. (a) proportional factors; (b) utilization; (c) allocated caps; (d) resource usage.

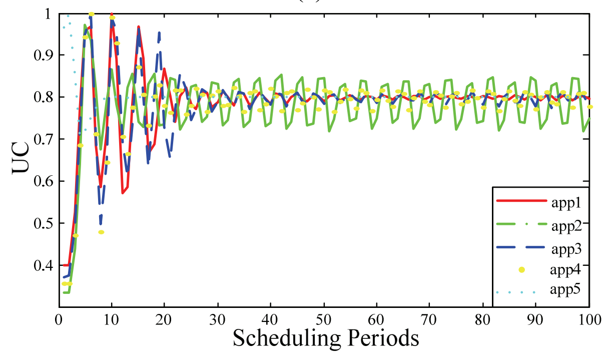
As shown in Fig. 9, during the first 30 experimental periods, results are the same as those included in Fig. 7. After 30 periods when available bandwidth jumps, our control method can react to bandwidth changes very fast. From the perspective of control theory, this variance in bandwidth can be considered as a disturbance, and Fig. 9 shows that the robustness, or anti-disturbance capability of our approach, is satisfactory.

D. *Parametric Convergence and Stability*

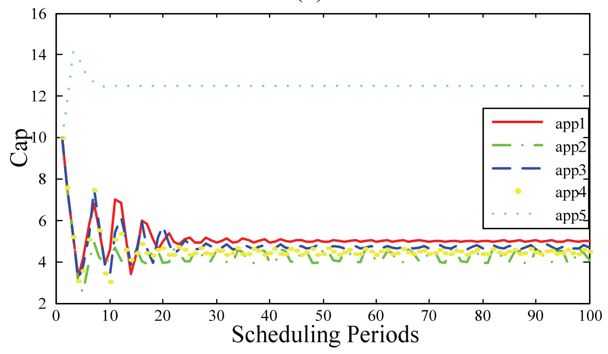
As a routine in control system design, stability analysis is indispensable because only stable systems can be applied. We find that system stability is sensitive to some parameters, e.g. *CapScale* in (7). A larger *CapScale* that results in a rapid convergence to a steady state may also lead to performance oscillation and instability. For example, when *CapScale* is set to 8, light applications will not converge to a stable value. In this case, the control system cannot work in a stable state, as shown in Fig. 10. The reason is that even the smallest control scale already exceeds required control power, so the system are adjusted either above or below and never reach the stable status.



(a)



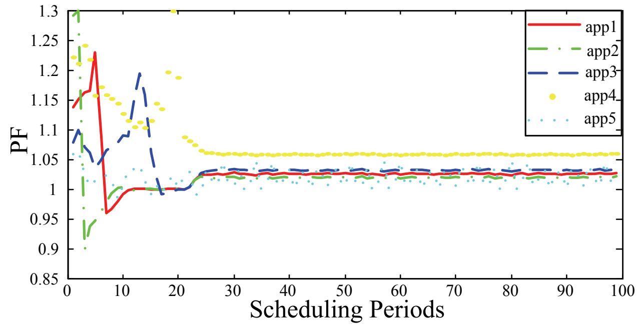
(b)



(c)

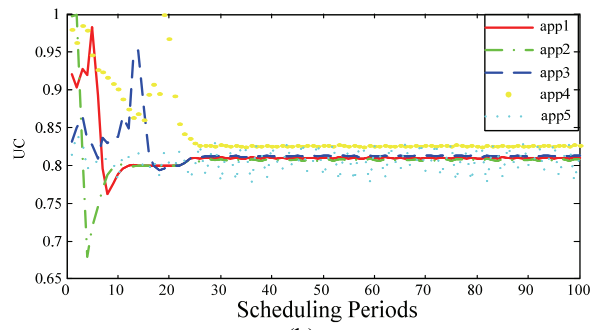
Fig. 10. Performance of light tasks with a large CapScale. (a) proportional factors; (b) utilization; (c) allocated caps.

Heavy tasks are not as sensitive to a large *CapScale* as light ones since their tolerance to the smallest *CapScale* is higher. In Fig. 11, *CapScale* is also set to 8. While some oscillation occurs, the magnitude is very small. In our applications we set *CapScale* from 3 to 5 based on our experience.

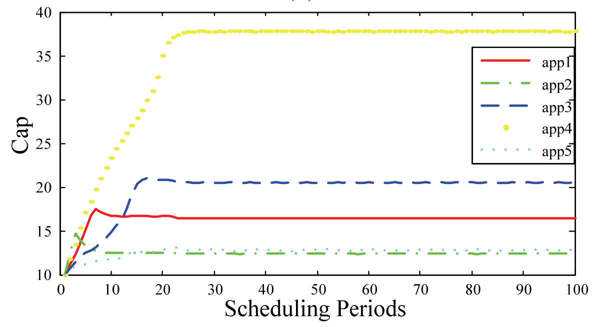


(a)

Fig. 11. (to be continued)



(b)



(c)

Fig. 11. Performance of heavy tasks with a large CapScale. (a) proportional factors; (b) utilization; (c) allocated caps.

E. Performance Comparison

Several other resource allocation schemes are developed in comparison, as shown in Table II: *iterative* and *even* stands for the way to allocate bandwidth among applications, in an iterative way as described in Section IV or just to divide the total bandwidth evenly; *dynamic* stands for CPU allocation manner described in Section III, while *fixed* means that allocated CPU resources for each application are constant. Obviously, our approach belongs to Case 1.

TABLE II. ALGORITHM SETTINGS

	Bandwidth	CPU
Case 1	*Iterative*	*dynamic*
Case 2	*Iterative*	*Fixed*
Case 3	*Even*	*dynamic*
Case 4	*Even*	*Fixed*

Some results are provided in Table III where performance metrics from top to bottom are final throughput (i.e., the volume of data processed during the experiment, and TP in short), CPU usage and bandwidth (BD in short) usage in percentage. Characters of H and L are abbreviations of *heavy* and *light*, indicating the type of applications. Still experiments are carried out for 100 periods and each period lasts for 100 seconds. Therefore, each scenario involves an upper limit of 50, 100 and 150Gb of data transfer, respectively, corresponding to three total bandwidth settings, i.e., 5, 10, and 15Mb.

Our algorithm prevails in all the scenarios in Table III. For example, using fewer resources, our algorithm obtains a higher throughput. In the extreme case, i.e., Case 4 where the bandwidth is evenly allocated and CPU resources are fixed, the performance is not satisfactory despite the fact that their resource consumption is substantial. The performance of Cases

2 and 3 is better than Case 4, but still not as good as ours. This result quantitatively validates our earlier assumption that unilateral adjustment of bandwidth or CPU resources is not sufficient enough to reach the goal of high throughput and high utilization of resources simultaneously. Furthermore it justifies our methodology to co-schedule bandwidth and CPU resources in an integrated and coordinated way.

TABLE III. PERFORMANCE COMPARISON

Index	Case		Total Bandwidth (Mbps)		
			5	10	15
Final TP (Mb)	H	1	**49951**	**97080**	**117140**
		2	46542	77107	77269
		3	46745	87495	104660
		4	48566	80004	82296
	L	1	**49999**	**99994**	**134980**
		2	46569	94019	123458
		3	47576	95018	124990
		4	48967	94948	104458
CPU Usage (%)	H	1	**40.17**	**77.76**	**98.81**
		2	50	50	50
		3	39.16	66.09	91.88
		4	50	50	50
	L	1	**9.79**	**20.74**	**31.93**
		2	50	50	50
		3	9.76	19.90	28.93
		4	50	50	50
BD Usage (%)	H	1	**93.49**	**91.50**	**81.77**
		2	93.08	77.11	51.51
		3	92.90	87.08	80.10
		4	90.13	80.00	54.86
	L	1	**92.14**	**92.02**	**89.99**
		2	93.14	93.14	90
		3	91.41	90.50	83.33
		4	93.24	91.99	80.33

VI. RELATED WORK

Stream processing [8] has been one of the major focuses of database research in recent years, and some techniques, e.g. Aurora [9] and TelegraphCQ [10], have been developed to deal with continuous queries on data streams. These systems often deal with light-weight data, i.e., the volume of each item in the incoming stream is small and usually in-memory processing is needed to yield an extremely low latency. Our work focuses on scheduling data streaming jobs on virtualized resources.

Distributed computing techniques evolve from cluster, grid to cloud computing [11]. Resource management and allocation has been a key issue in these areas [12]. In cloud computing with virtualization technology as the key enabler, virtual machines [13] or virtual clusters [14] are basic units in management, scheduling and optimization [15]. Tools including Eucalyptus [16], VMPlants [17] and Usher [18] can serve this management purpose. Some schedulers are developed to support data streaming applications, e.g. GATES [19] and Streamline [20], but they mainly concern on computing resource allocation without taking bandwidth into account. Several projects such as EnLIGHTened [21], G-lambda [22] and PHOSPHORUS [23] put emphases on networking resources. They have the total control over a dedicated optical network so that a deterministic path can be obtained with advance reservation or on demand, while our work uses a TCP/IP based shared network, such that bandwidth is more of an issue. Streamflow [40] proposes a framework that enhances a traditional scientific workflow infrastructure to enable the stream processing capability, and resource allocation is not discussed. In [41] a distributed and pipelined dataflow execution system is proposed. The execution is optimized by exploiting parallelism, load balancing and co-locating. In our approach all the execution is in a virtual environment and we have a unique requirement to harmonize computation and bandwidth allocation.

Control theory has been successfully applied to optimize performance or quality of service (QoS) for various computing systems. An extensive summary of related work can be found in the first chapter of this book [7]. Approaches such as proportional, integral, and derivative (PID) control [24][25], pole placement [26], linear quadratic regulator (LQR) [26] and adaptive control [27][28] are applied in various systems. Most of them require a precise model of the controlled object(s). Using these approaches it is inevitable to establish differential equation models that define the system response to inputs. This is rather challenging in some cases including data streaming applications, due to variable coupling and heavily nonlinear property of the system. Fortunately, fuzzy control offers an alternative, providing a formal technique to represent, manipulate and implement human experts' heuristic knowledge for controlling an object via IF-THEN rules. Fuzzy control does not rely on mathematical modeling of an object and instead it establishes a direct nonlinear mapping between inputs and outputs. This can significantly reduces the difficulty of a control system's design in our application scenario.

The first application of fuzzy control was introduced into industry in 1974 [29]. Fuzzy control [30][31] is also a research topic in computing systems mainly on admission control to get a better quality of service. Adaptive fuzzy control is applied for utilization management of periodic tasks [32], where the utilization is defined as the ratio of the estimated execution time to the task period. Fuzzy inference is carried out to decide the threshold over which the QoS of tasks should be degraded or even incoming tasks be rejected. In this work, execution time estimation must be provided, which is not feasible for some applications.

A recent study relevant to our work [33] focused on providing predictable execution so as to meet the deadlines of tasks. Virtualization technology is applied to implement the so-called performance container and compute throttling framework, to realize the "controlled time-sharing" of high performance computing resources. System identification is carried out to establish the model of controlled object and a proportional and integral (PI) controller is used. This work has similar motivation with ours, while we adopt the model-free fuzzy control approach given the nature of the considered data streaming applications.

VII. Conclusion

In this work we provide a new approach for co-scheduling and co-allocation of virtualized resources for data streaming and processing. Fuzzy logic control of CPU resources and iterative bandwidth allocation are implemented for integrated and coordinated resource management. Experimental results show the good performance in resource utilization, data throughput and robustness of the approach, in comparison with several other methods with less adaptability to dynamic environments.

In further work, besides CPU and bandwidth, we plan to investigate virtualized network and elastic storage allocation. More complex scenarios, such as workflows [34][42], service composition [35], and economical provisioning [47] will be considered. In these scenarios VMs can be used for each stage in a process and optimized to achieve a balanced resource allocation and thus a better overall performance.

Virtualization technology has been applied in the LIGO community for data analysis software development [45]. In future virtualization based dynamic control will become more crucial for larger-scale runtime LIGO data streaming and processing, since the development of next generation of gravitational wave detectors [46] will involve much larger amount of data and require more computation and network resources.

Acknowledgment

We thank the anonymous reviewers for detailed and constructive comments. This work is carried out in close collaboration with LIGO (Laser Interferometer Gravitational-wave Observatory) Laboratory at Massachusetts Institute of Technology. Junwei Cao would like to express his gratitude to Dr. Erik Katsavounidis for his long-term support on LIGO gravitational wave data analysis work at Tsinghua University.

References

[1] W. Zhang, J. Cao, Y. Zhong, L. Liu, and C. Wu, “Grid Resource Management and Scheduling for Data Streaming Applications”, *Computing and Informatics*, Vol. 29, pp. 1001-1028, 2010.

[2] A. Abramovici, W. E. Althouse, et. al., “LIGO: The Laser Interferometer Gravitational-Wave Observatory”, *Science*, Vol. 256, No. 5055, pp. 325 – 333, 1992.

[3] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, “A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis”, in I. J. Taylor, D. Gannon, E. Deelman, and M. S. Shields (Eds.), *Workflows for eScience: Scientific Workflows for Grids*, Springer Verlag, pp. 39-59, 2007.

[4] M. Migliardi, J. Dongarra, A. Geist and V. Sunderam, “Dynamic Reconfiguration and Virtual Machine Management in the Harness Metacomputing System”, *Computing in Object-Oriented Parallel Environments*, Lecture Notes in Computer Science, Vol. 1505, pp. 127-134, 1998.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T.L. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, “Xen and the Art of Virtualization”, *Proc. ACM Symp. on Operating Systems Principles*, 2003.

[6] W. Zhang, J. Cao, Y. Zhong, L. Liu and C. Wu, “An Integrated Resource Management and Scheduling System for Grid Data Streaming Applications”, *Proc. 9th IEEE/ACM Int. Conf. on Grid Computing*, pp. 258-265, Tsukuba, Japan.

[7] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*, Wiley-IEEE Press, August 2004.

[8] L. Golab, and M. T. Ozsu, “Issues in Data Stream Management”, *SIGMOD Record*, Vol. 32, No. 2, pp. 5-14, 2003.

[9] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, “Aurora: A New Model and Architecture for Data Stream Management”, *VLDB Journal*, Vol. 12, No. 2, pp. 120-139, 2003.

[10] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, “TelegraphCQ: Continuous Dataflow Processing”, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[11] I. Foster, Y. Zhao, I. Raicu, S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared”, *Proc. IEEE Grid Computing Environments, conj. IEEE/ACM Supercomputing Conf.*, Austin, 2008.

[12] R. Buyya, C. S. Yeo, S. Venugopala, J. Broberga, and I. Brandicc, “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility”, *Future Generation Computer Systems*, Vol. 25, No. 6, pp. 599-616, 2009.

[13] M. Rosenblum and T. Garfinkel, “Virtual Machine Monitors: Current Technology and Future Trends”, *IEEE Computer*, Vol. 38, No. 5, pp. 39-47, 2005.

[14] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayer and X. Zhang, “Virtual Clusters for Grid Communities”, *Proc. IEEE Int. Symp. on Cluster Computing and the Grid*, May 2006.

[15] F. Zhang, J. Cao, H. Cai, L. Liu, and C. Wu, “Redundant Virtual Machines Management in Virtualized Cloud Platform”, *Int. J. Modeling, Simulation, and Scientific Computing*, Vol. 2, No. 2, pp. 151-168, 2011.

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus Open-Source Cloud-Computing System”, *Proc. 9th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, 2008.

[17] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, “VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing”, *Proc. ACM/IEEE SC2004 Conference*, 2004.

[18] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, “Usher: an Extensible Framework for Managing Clusters of Virtual Machines”, *Proc. 21st Conf. on Large Installation System Administration*, 2007.

[19] L. Chen and G. Agrawal, “A Static Resource Allocation Framework for Grid-based Streaming Applications”, *Concurrency and Computation: Practice and Experience*, 18:653–666, 2006.

[20] B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, “Streamline: Scheduling Streaming Applications in a Wide Area Environment”, *Multimedia Systems*, Vol. 13, No. 1, pp. 69-85, 2007.

[21] L. Battestilli, et al., “EnLIGHTened Computing: An Architecture for Co-allocating Network, Compute, and Other Grid Resources for High-End Applications”, *Proc. Int. Symp. on High Capacity Optical Networks and Enabling Technologies*, pp. 1-8, 2007.

[22] A. Takefusa, et al., “G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS”, *Future Generation Computer Systems*, Vol. 22, No. 8, pp. 868-875, October 2006.

[23] S. Figuerola, et al., “PHOSPHORUS: Single-step On-demand Services across Multi-domain Networks for e-Science”, *Proc. SPIE*, Vol. 6784, 67842X, 2007.

[24] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, “Performance Guarantees for Web Server End-systems: A Control-theoretical Approach”, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, 2002.

[25] S. Parekh, N. Gandhi, J. L. Hellerstein, D. Tilbury, T. S. Jayram, and J. Bigus, “Using Control Theory to Achieve Service Level Objectives in Performance Management”, *Real Time Systems Journal*, Vol. 23, No. 1-2, 2002.

[26] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D.M. Tilbury, “MIMO Control of an Apache Web Server: Modeling and Controller Design”, *Proc. American Control Conf.*, 2002.

[27] A. Kamra, V. Misra, and E. M. Nahum, “Yaksha: A Self-tuning Controller for Managing the Performance of 3-tiered Web Sites”, *Proc. 12th IEEE Int. Workshop on Quality of Service*, June, 2004.

[28] Y. Lu, C. Lu, T. Abdelzaher, and G. Tao, "An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services", *Proc. 10th IEEE Int. Workshop on Quality of Service*, May 2002.

[29] P. J. King and E. H. Mamdani, "Application of Fuzzy Algorithms for Control Simple Dynamic Plant", *IEE Proc. Control Theory App.*, Vol. 121, pp. 1585-1588, 1974.

[30] Y. Diao, J. L. Hellerstein, S. Parekh, "Using Fuzzy Control to Maximize Profits in Service Level Management", *IBM Systems Journal*, Vol. 41, No. 3, 2002.

[31] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations", *Communications of ACM*, Vol. 17, pp. 1632-1650, 1999.

[32] M. H. Suzer and K. D. Kang, "Adaptive Fuzzy Control for Utilization Management", *Proc. IEEE Int. Symp. on Object/Component/ Service-oriented Real-time Distributed Computing*, 2008.

[33] S. Park and M. Humphrey, "Feedback-controlled Resource Sharing for Predictable eScience", *Proc. ACM/IEEE Conf. on Supercomputing* 2008.

[34] R. Prodan, M. Wieczorek, "Bi-Criteria Scheduling of Scientific Grid Workflows", *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 2, pp. 364-376, 2010.

[35] W. Tan, Y. Fan, M. Zhou, and Z. Tian, "Data-driven Service Composition in Building SOA Solutions: A Petri Net Approach", *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 3, pp. 686-694, 2010.

[36] The Xen Hypervisor. http://xen.org/

[37] W. Pedrycz. 1993. *Fuzzy Control and Fuzzy Systems* (2nd ed.). John Wiley & Sons, Inc., New York, NY, USA.

[38] J. Cao and J. Li, "Real-time Gravitational-wave Burst Search for Multi-messenger Astronomy", *Int. J. Modern Physics D*, Vol. 20, No. 10, pp. 2039-2042, 2011.

[39] S. Pandey, "Scheduling and Management of Data Intensive Application Workflows in Grid and Cloud Computing Environments", *Doctor of Philosophy Thesis, The University of Melbourne*, Australia, 2010.

[40] C. Herath, and B. Plale, "Streamflow Programming Model for Data Streaming in Scientific Workflows," *Proc. 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, pp. 302-311, 2010.

[41] C. S. Liew, M. P. Atkinson, J. I. v. Hemert et al., "Towards Optimising Distributed Data Streaming Graphs using Parallel Streams," *Proc. 19th ACM Int. Symp. on High Performance Distributed Computing*, Chicago, Illinois, pp. 725-736, 2010.

[42] D. Zinn, Q. Hart, T. McPhillips et al., "Towards Reliable, Performant Workflows for Streaming-Applications on Cloud Platforms," *Proc. 11th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing*, pp. 235-244, 2011.

[43] J. Cao, E. Katsavounidis, and J. Zweizig, "Grid Enabled LIGO Data Monitoring", *Proc. IEEE/ACM Supercomputing Conf.*, Seattle, WA, USA, 2005.

[44] R. Pordes for the Open Science Grid Consortium, "The Open Science Grid", *Proc. Computing in High Energy and Nuclear Physics Conf.*, Interlaken, Switzerland, 2004.

[45] W. Chen, J. Cao, and Z. Li, "Customized Virtual Machines for Software Provisioning in Scientific Clouds", *Proc. 2nd Int. Conf. on Networking and Distributed Computing*, Beijing, China, 2011.

[46] LIGO Scientific Collaboration, "A Gravitational Wave Observatory Operating beyond the Quantum Shot-noise Limit", *Nature Physics*, Advance Online Publication, 2011.

[47] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe and C. Pu, "Economical and Robust Provisioning of N-Tier Cloud Workloads: A Multi-level Control Approach," *Proc. IEEE International Conference On Distributed Computing Systems (ICDCS)*, Minneapolis, Minnesota, USA, 2011.

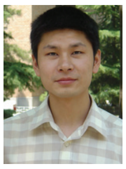
BIOGRAPHIES

Junwei Cao (M'99–SM'05) received his Ph.D. in computer science from the University of Warwick, Coventry, UK, in 2001. He received his bachelor and master degrees in control theories and engineering in 1998 and 1996, respectively, both from Tsinghua University, Beijing, China.

He is currently a Professor and Vice Director, Research Institute of Information Technology, Tsinghua University, Beijing, China. He is also with Tsinghua National Laboratory for Information Science and Technology, Beijing, China. He is now a Visiting Scientist of MIT LIGO Laboratory. Before joining Tsinghua University in 2006, he was a Research Scientist at MIT LIGO Laboratory and NEC Laboratories Europe for about 5 years. He has published over 130 papers and cited by international scholars for over 2200 times. He is the book editor of Cyberinfrastructure Technologies and Applications, published by Nova Science in 2009. His research is focused on advanced computing technologies and applications.

Prof. Cao is a Senior Member of the IEEE Computer Society and a Member of the ACM and CCF. He is also a member of LIGO Scientific Collaboration.

Wen Zhang (M'07) works in Chongqing Military Delegate Bureau, General Armament Department of PLA. He received his PhD in control engineering and applications from Department of Automation, Tsinghua University, Beijing, China, in 2010. His research was focused on dynamic control of data streaming and processing.



Wei Tan received the B.S. degree and the Ph.D. degree in Control Science and Engineering, from Department of Automation, Tsinghua University, China in 2002 and 2008, respectively. He is currently a research staff member in IBM T. J. Watson Research Center, USA. From 2008 to 2010 he was a researcher at Computation Institute, University of Chicago and Argonne National Laboratory, USA. His research interests include service-oriented architecture, business and scientific workflows, Petri nets, workload optimization, business intelligence and data centric computing. He has published over 30 journal and conference papers. He is Associate Editor of the IEEE Transactions on Automation Science and Engineering. He served in program committee of many international conferences and was the PC co-chair of the First IEEE/ACM Workshop on the application of Social Networking concepts to Cluster, Cloud, Grid and Services Computing (SN4CCGridS). Contact him at wtan@us.ibm.com.