Large In-Memory Cyber-Physical Security-Related Analytics via Scalable Coherent Shared Memory Architectures

John R. Williams, Sergio Herrero, Christopher Leonardi, Stephen Chan, Abel Sanchez, Zeyar Aung
Engineering Systems Division,
Massachusetts Institute of Technology
Cambridge USA
jrw@mit.edu

***Abstract*—Cyber-physical security-related queries and analytics run on traditional relational databases can take many hours to return. Furthermore, programming analytics on distributed databases requires great skill, and there is a shortage of such talent worldwide. In this talk on computational intelligence within cyber security, we will review developments of processing large datasets in-memory using a coherent shared memory approach. The coherent shared memory approach allows programmers to view a cluster of servers as a system with a single large RAM. By hiding the actual system architecture under a software layer, we proffer a more intuitive programming model. Furthermore, the design of applications is "timeless" since hardware upgrades require no changes to the software. The advantages of shared memory are countered by some disadvantages in that race conditions can occur; however, in many of these cases, we can provide models that protect us against such problems. Exemplars include sensemaking of Twitter feeds, the processing of Smart Meter datasets, and the large scale simulation of the caching of files at disparate points around the globe.** *(Abstract)*

Keywords-sensemaking; multicore; coherent shared memory; (key words)

I. INTRODUCTION

Dynamic global supply chains and the reliance on geospatial decision support systems (DSS) have underscored the need to quickly analyze large datasets for rapid sensemaking, involving cyber-physical security-related queries and analytics. While many enterprise systems were principally architected and optimized for performance, next-generation intelligent engineering system approaches necessitate architecting for robust security so that the system is evolutionarily resilient and defensible at machine-speed. These considerations have necessitated a paradigm shift in dataset/computation algorithms; in a predominant share of big data cases, moving the computation to the data instead of the traditional methodology of moving the data to computation. The processing speed advantage can be several orders of magnitude, which immediately alters the outcomes of numerous cyber security cost-benefit analysis (CBA) models. The prior ROI conclusions that certain cyber security technologies may hinder productivity are reversed, and an entire spectrum of cyber defense-in-depth approaches now come into play and create new opportunities.

As an exemplar, the all-important human component within DSS leads to the conclusion that limited human cycles necessitates careful prioritization in dealing with the Pandora's Box of cyber security concerns. However, with the promise of processing large datasets in-memory, via the movement of the computation to the dataset, utilizing a coherent shared memory (CSM) approach, a more intuitive programming model allows the less expert multicore/multi-threaded programmer to engage in the cyber assurance arena. With this ability to view a high performance computing (HPC) cluster of servers as an integrated system with a single large random-access memory (RAM), information assurance project managers are now able to scale and bring to bear a legion of non-specialized programmers. Certain mission-critical cyber security domains can now receive an infusion of much-needed human cycles. The CSM approach serves as a launching pad to propel cyber practitioners towards instantiating new participatory and collaboration mechanisms among non-specialized programmers that might allow us to address the deluge of sensor data from the "Internet of Things" (IOT) and other sources.

II. ANALYTICS AND THE SECURITY PROBLEM

In many of the time-sensitive analytics cases, real-world applications require algorithms that can return within minutes rather than hours; intrinsically, they rely upon the aforementioned sensemaking to locate the pertinent information over very large data volumes in quasi-real time. Furthermore, with every new observation from the myriad of IOT artificial/human sensors, these sensemaking algorithms, which take advantage of CSM, can readily be configured to conjoin diverse data sets and ascertain non-evident relationships for expressive context accumulation (ECA). Sensemaking may engage in provenance/pedigree resolution

978-1-4244-9906-9/11/$26.00 ©2011 IEEE

and data tethering to distinguish noise (e.g. re-Tweets) for robust semantic reconciliation (RSR), and reevaluating prior heuristical assumptions, so as to recalculate for knowledge management actualization sequence neutrality (KMASN).This triumvirate of ECA, RSR, and KMASN allows us to effectuate a new model of computational intelligence within cyber security by reversing the conventional and prototypical notion that the more big data, the slower the involved system. Extrapolating upon the learning's of multiplier recodings, which exhibit a performance increase with a higher implementation of zeroes, we are able to harness faster algorithms with big data. Future work based on our prior art, such as the simulation of future systems to benchmark the performance of central indexing and file caching, et al, with respect to moving target defense (MTD) paradigms, will serve as important intellectual and operational exercises for tapping into CSM's inherent advantage of obfuscating the actual system architecture under a software layer. This will allow us to simultaneously decrease the known attack surface area while taking simultaneous advantage of species diversification to avoid transitive closure.

III. Real time feeds - Twitter

In spite of its simplicity, Twitter has rapidly become a powerful communication medium that is used by millions of people every day. It enables users to broadcast short messages, called tweets, to the public through the personal feed exposed in the Twitter site, or more effectively, to other users subscribed to them through a social networking platform of the service. Twitter's main advantage is that the messages are broadcasted instantaneously, as opposed to other platforms in which information is pulled from servers by subscribed clients. Even though tweets are restricted to 140 characters, in general, they contain relatively well-formed succinct pieces of information that are optionally accompanied by timestamps, geotags and URLs. Its public nature, combined with its real-time broadcasting characteristics, bring to the service the potential of discovering what is happening at any point in time. Since Twitter's inception, multiple ways of exploiting the service have been devised: Initially created for users to make

Figure 1: Needle in the Haystack

their ideas public, slowly has been taken over by advertising agencies to publicize products or services, celebrities to influence their fans or political movements to promote ideas. Unfortunately, bots posting junk messages have taken possession of the platform as well. Extracting useful information from Twitter is known as the "Needle in the Haystack" problem, which is illustrated in Figure 1. In this section a technique to filter tweets related to music events in NY is analyzed. Nevertheless, the same principles can be applied to cyber-security related events.

For this purpose, the public feed of tweets generated from September 16, 2009 to October 10, 2009 was collected. Prior to proceed to the construction of the feature vector that will enable the classification of tweets, it was necessary to preprocess the tweets facilitate the experiment.

A. *Tweet Preprocessing*

1) Language Selection: For the purpose of this analysis we constraint the search space to the tweets written in English. The extraction of tweets through the Twitter API provides information about the language of the site that the Twitter user utilized to post the tweet [1]. However, this does not guarantee that the content of the tweet will be written in that language. In order to reduce the space only to tweets written in English, tweets containing 4 or more words from the American English dictionary were considered. These method yields an accuracy of 99% of tweets consistently, although cases in which a single tweet contained a fragment in English and another fragment on different languages were observed.

2) Geolocation: In order to experimentally test the validity of the approach of this paper, it was necessary to reduce the tweet space to a geographical area with the following requirements:

- High Activity: Although the growth on activity in Twitter has been noticeable for the last two years, the distribution of locations in which the service is used is highly discontinuous and concentrated primarily in urban areas, and in specific countries. According to World of Tweets [2], US has generated 36% of the tweets and a large percentage of these came from in New York metropolitan area.
- Popular Event: In order to have an environment in which our proposed techniques could be tested, it was necessary to have a noticeable Twitter activity on the event of interest. For the case of music events, New York metropolitan area is an appropriate location known for its concentration of events.

Alternative Event Sources: Our technique is reinforced with prior information about the events that will take place in the chosen location. It is important to have alternative sources of information about the event available for processing, so as use this information to help our technique. Again, New York provides these sources, for this project NYC.com was used [3].

The reduction of the search space was carried out by selecting those tweets explicitly containing New York as their location of origin, or in case of containing lat-long coordinates, it was required to fall into a radius of 20 km from Manhattan (40◦43'42" N, 73◦59'39" W).

B. Part-Of-Speech Tagging of Tweets

Prior to design the feature vector that will be used for classification, it is necessary to add the Part-Of-Speech (POS) tags to the tweets. This additional information will be used later in order to create the list of keywords. In general, Tweets are well-formed sentences with enough information to be tagged accurately. For this project, the Stanford Log-linear Part-Of-Speech Tagger was used [4].

C. Construction of Feature Vector

In order to extract the keywords that will generate the definition of the feature vector, 2000 tweets were manually labeled, discerning tweets related to music events (Concert label) from the rest of tweets (Background label). 1840 (92%) were labeled as Background, while 160 (8%) were labeled as Concert. Then, each of these 2000 tweets was passed to the Part-Of-Speech tagger in order to generate the POS tag information. We focus particularly on the following tags: 'NN' (Noun, singular common), 'NNP' (Noun, singular proper), 'NNS' (Noun, plural common), 'NNPS' (Noun, plural proper), 'VB' (Verb, base form), 'VBD' (Verb, past tense), 'VBG' (Verb, present participle), 'VBN' (Verb, past participle), 'VBP' (Verb, present tense, not 3rd person singular) and 'VBZ' (Verb, present tense, 3rd person singular). We will call this subset of POS tags, the *Relevant Tag Set*. Given the words, their tags and their labels corresponding to each of these tweets, two groups were generated:

1) Group 1: The collection of unique words that belong to tweets labeled as 'Concert' and are tagged with tags in the *Relevant Tag Set*.

2) Group 2: The collection of unique words that belong to tweets labeled as 'Background' and are tagged with tags in the *Relevant Tag Set*.

The construction of these two groups is carried out under the premise that nouns and verbs convey the largest amount of contextual information on a well-formed sentence. Group 1 contains a collection of nouns, such as Jazz, Theater, Concert or Album, and verbs, such as listen, play or sing, that are specific for tweets related to music events in New York. Group 2 will contain nouns, such as New York, Manhattan or night and verbs, such as "to be", "to have" or "to go" that are not necessarily related to music events. These common nouns and verbs can also appear in Group 1. For this reason, it is necessary to remove this 'noise' from Group 1 in order to come up with a representative keyword list composed by nouns and verbs strongly associated to music events in New York. This is done by removing the most popular words in Group 2 from Group 1; a cut-off count parameter τ is established. For this project we set $\tau = 50$. Therefore, all words that appear more than τ times in Group 2 will be removed from Group 1. The keyword list is then used to construct the feature vector $\bar{x}$ for each tweet. d is the number of unique keywords after removing the noise from Group 1. Features have the following form:

$$x_i = f_i(w, y) = \begin{cases} 1, & for\ w = w_i\ and\ y = 1 \\ 0, & otherwise \end{cases}$$

where $i = 1 \ldots d$.

D. Augmented Feature Vector

So far only information obtained from the Twitter platform was utilized to construct the feature vector. In this subsection, the possibility of including additional information from non-twitter sources is contemplated. For the specific case of events in New York, a detailed list of events for the time window considered was downloaded from *NYC.com*. This list consisted of title, location, time, description and ticket price information about individual music events. Analogous to section III-B, the POS tagger was utilized to obtain the tags for the title and description of the event. Given the words and tags for each event, a list with unique words, called Group 3, tagged as with tags in the *Relevant Tag Set* was constructed. Finally, the most popular words in Group 2 were removed from Group 3 to yield a list of additional keywords aiming to augment our initial feature vector. This additional list of keywords is then merged with the original list of keywords to come up with an "augmented" feature vector. The construction of the augmented feature vector is illustrated in Figure 2. Next, we proceed to present the performance results obtained by using these two versions of the feature vector to classify tweets.

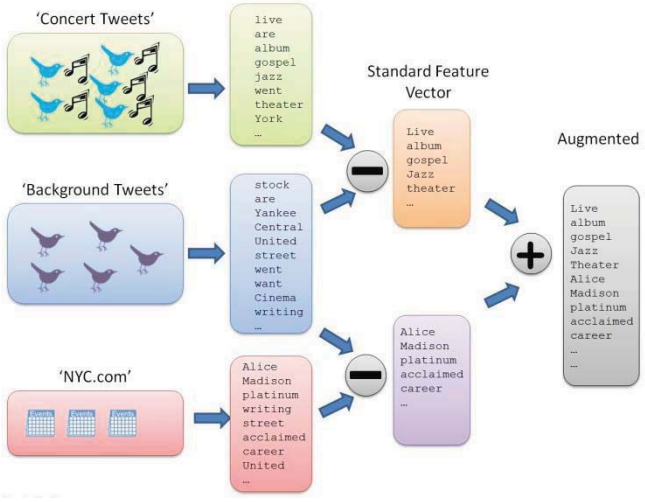

Figure 2: Augmented Feature Vector

E. Classification Performance

After these two definitions of feature vectors, the standard and the augmented, have been defined we proceed to evaluate the classification performance. In this section a Soft-Margin Support Vector Machine (SVM) classifier was utilized. The collection of tweets was randomly divided into two sets, the training set 80% and the testing set 20%. Different kernels were tested, Linear, Polynomial and Radial Basis function (RBF); the RBF kernel yield the best performance results for a regularization parameter C = 10 and γ =1.0.

Table 1: Classification Results

(standard, augmented)	Predicted 'Background'	Predicted 'Concert'
'Background'	(363,363)	(3,2)
'Concert'	(9,0)	(25,35)

Given that 92% of the tweets belong to the 'Background' class, the results obtained by the SVM classifier are required to improve this value. Table 1 presents the results obtained by the standard and augmented feature vectors: The standard feature vector resulted on an accuracy of 97%. The additional information provided by the event database raised the classifier performance to a 99% for the augmented feature vector, from 97% consistently for randomized runs of the same dataset.

IV. Large Data Analytics and Storage Strategies for Smart Meter Data Processing

A number of governments and organizations around the world agree that the first step to address national and international problems, such as energy independence, global warming or emergency resilience, is the redesign of electricity networks, known as Smart Grids. Typically, power grids have 'broadcasted' power from generation plants to large population of consumers on a suboptimal way. Nevertheless, the fusion of energy delivery networks and digital information networks, along with the introduction of intelligent monitoring systems (Smart Meters) and renewable energies, would enable two-way electricity trading relationships between electricity suppliers and electricity consumers. The availability of real time information on electricity demand and pricing would enable suppliers to optimize their delivery systems, while consumers would have the means to minimize their bill by turning on appliances at off-peak hours. The construction of the Smart Grid entails the design and deployment of information networks and systems of unprecedented requirements on storage, real-time event processing and availability. In order to identify the solution that meets these requirements, a simulator capable of reproducing the smart meter load of a city of 1 million households was constructed. The generated dataset was stored on a coherent shared-memory architecture. The decomposition of dynamic pricing mechanisms into MapReduce primitives, and their execution in this type of system produced an order of magnitude of acceleration compared to classic relational database systems.

Currently, the design and size of an electricity supply network corresponds to the demand observed during peak periods. Unfortunately, this leads to an over dimensioned network for the rest of the off-peak periods. Therefore, one of the *Smart Grid*'s goals is to modify customer behavior patterns in order to balance the load and reduce peak electricity demand. A naive way to motivate this behavior shift is to apply tariffs based on the time of usage, setting different electricity prices for peak and off-peak periods. Even the simplest pricing mechanism requires information about the time and volume of electricity consumed by every customer. For this purpose, *Smart Meters* (SMs) will be installed in every household participating in the *Smart Grid*. These devices will periodically measure and temporarily store each customer's consumption data, and then this information will be transferred to the utility companies through the Advanced Metering Infrastructure (AMI) [5]. Here we present a system architecture that aims to provide a solid foundation for the data consumption systems that will make the *Smart Grid* an intelligent network.

A. Introduction

A city such as Los Angeles, with over 10 million meters, will generate over 1 billion reads per day. Thus, a monthly bill would require processing of 30 billion reads. Here we investigate the architecture that would allow producing the monthly bill for the entire city on a timely manner. The components of the Smart Grid architecture are summarized below.

Concentrator node (CN): A Concentrator node gathers, stores and returns electricity consumption data from multiple SMs. It is a passive: it receives and executes orders. At the end of the day, the CN is asked to collect the SM readings from each household linked to it and stores the data so that it can be processed later on demand. The communication protocols between the Concentrator Node and the SMs are specified by AMI [5].

Central Data Processing node (CDPN): The Central Data Processing node manages the CNs. As the only active node in the system, it constitutes the highest level of control. It is responsible for managing and coordinating the tasks assigned to CN as well as calculating electricity consumption statistics and monthly billing. Besides, the CDPN generates a set of *Time Buckets* to calculate the monthly bill. A *Time Bucket* is defined as "a continuous or intermittent period of time in which all the SM readings have the same price". An example of a *Time Bucket* is the following: "Saturdays and Sundays, from 0:00AM to 5:59AM between January 1, 2009 and January 31, 2009".

B. Hybrid Storage (RDBMS and FS)

Next, a storage approach is proposed based on the combination of a File System (FS) and a Relational Database Management System (RDBMS). As it is illustrated in Figure 3, this architecture is composed by one CDPN with a single database and a set of CN's equipped with their local File Systems. In this approach the Smart Meter reading information is not stored in the database, but in XML files in the FS instead. The database stores metadata and pointers to these files. The database consists of a single table and two columns: one with the pointer to the XML file containing the timestamp information and the other with the pointer to the XML file containing the electricity consumption.

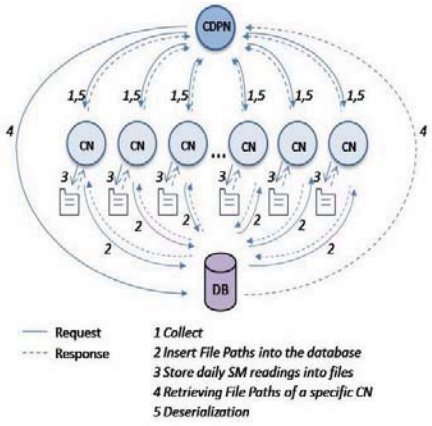

Figure 3: Hybrid storage, database and file system

We used the local File System for convenience; nevertheless a Distributed File System (DFS) could also be used for this architecture. Every day CDPN asks each CN to collect the data from each household. Finally, at the end of the month the CDPN loads all the data into coherent memory and executes the in-memory *Multi-Core Billing (MCB)* algorithm. Next the design of the MCB algorithm is explained.

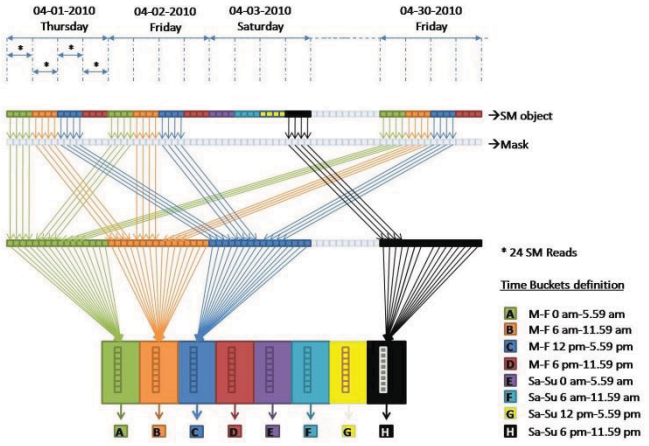

Figure 4: In-Memory Multi-Core Billing (MCB) Algorithm

In-memory Multicore Billing (MCB) Algorithm: An inherent advantage of storing the electricity consumption information in-memory at the CDPN node is the possibility of utilizing state-of-the-art multicore processors to accelerate data processing. In this project, an algorithm that aggregates Smart Meter reading data into *Time Buckets* was developed. For each monthly collection of readings, each read is classified to the corresponding bucket and then the Smart Meter readings within the same *Time Bucket* are aggregated. The algorithm works as follows: The entire dataset is divided by household into *N* partitions, where *N* is the number of processing cores available, and then each core receives one of these partitions. For each household, the thread assigned to the processing core follows a two-step process.

1) *Sort Phase*: The thread sorts the Smart Meter readings so that those that belong to the same *Time Bucket* are contiguous in memory.

2) *Aggregate Phase*: The thread takes the sorted values for each bucket and carries out a reduction operation. The reduction of data residing in contiguous locations of the memory is an extremely fast operation due to data pre-fetching.

In order to accelerate the sorting phase the use of expensive conditional statements (if) is avoided, and an indirection array is introduced instead. The indirection array, called *Mask*, contains the index of the location of each SM in the sorted array. The *Mask* is associated to each *Time Bucket* definition and is pre-calculated once for all the Smart Meters. The algorithm is illustrated in Figure 4.

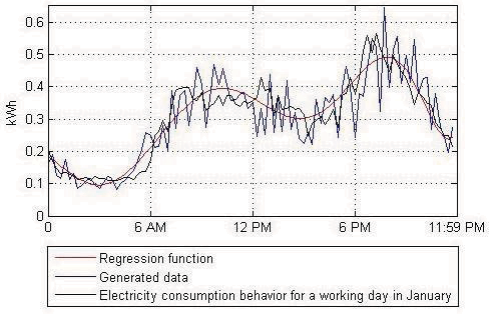

Figure 5: Generation of one household's daily SM Readings

C. Performance Results

This section presents the results obtained during the simulation phase. Every simulation stores and processes Smart Meter reading data of an entire month (31 days) for a specified number of households per CN node. The amount of data that a CN node has to handle per household is 2976 Smart Meter readings/month (96 readings/day) as Smart Meter readings are defined to indicate electricity consumption during 15 min

intervals. Every bill calculation is based on the collection of 8 *Time Buckets* described in Figure 4. It is important to indicate that *Time Bucket* definitions are not available until the month has concluded; therefore, pre-aggregation is not an option in this study.

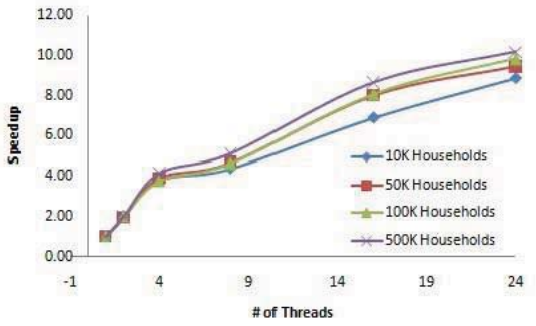

Figure 6: Multi-Core Billing (MCB) Algorithm's performance analysis

The hybrid architecture was tested considering one CDPN node and ten CN nodes with 100,000 households each. It took ~2 min to write the CN's collected daily electricity data into the File System. Additionally, the process of de-serializing all monthly data from disk on a "cold start" was measured, and resulted in approximately 340 min. Nevertheless, this can be avoided if data is kept in coherent memory. Finally, we explored the scalability of our *Multi-Core Billing (MCB)* algorithm. We carried out experiments for different numbers of households and different numbers of threads. Figure 6 describes the speedup produced by the use of multicore technology. From these results, we concluded that the hybrid RDBMS and File System solution resulted on better scalability and performance on the storage of Smart Meter reading information. These characteristics are critical to ensure the security of the energy grid. We showed the benefits of processing Smart Meter reading data in-memory using state-of-the-art multicore processors. Unlike batch processing systems, such as Hadoop, in-memory multicore processing can accelerate the calculation of prices and readjustments of tariffs and help to approach the real-time goal of the *Smart Grid*. We have also shown that storing the data in text files, while keeping the database for metadata in these files, provide horizontal scalability for the system, which cannot be achieved by classic RDBMS solutions, which only scale to tens of nodes.

V. Global Data Infrastructure Simulator

In this section, the construction of a Global Data Infrastructure simulator running on coherent shared memory architectures is presented. The goal of the simulator is to evaluate the impact of "what if" scenarios on the performance, availability and reliability of the overall system. The simulator takes as input the workload of each application, the resources allocated by individual user requests, the network topology of the organization, the hardware configuration deployed in each datacenter and details on background processes. Using this information, the set of models that integrate the simulator produce estimates of the response time for each request and user, along with measurements of resource allocation and network utilization. This simulator has a direct application on the testing of mechanisms to defend infrastructures against cyber-attacks.

A. Multi-Agent System (MAS)

The simulator platform is constructed following a multi-layered approach formed by *Components* and *Operations*.

Components are stateful autonomous agents that represent parts of the infrastructure at various granularities. These components interact with each other through message exchange: They accept input messages of pre-established types that alter their internal states, and based on this state they produce output messages addressed to other components. High-level components can stand for datacenters or network links. These include medium-level components such as application server tiers or database server tiers. Servers, in turn, are modeled using low-level components such as memory, processors, disk arrays or network cards. The modularity of this design grants the flexibility to add/remove or reutilize components. Specific behaviors associated to each type of component can be adjusted to the real hardware by tuning its configuration parameters. These behaviors are modeled using queuing network theory. A basic example of a global IT infrastructure with different component granularities is illustrated in Figure 7. This infrastructure is composed by a master datacenter component, D_{NA} , and multiple slave datacenter components (D_{EU}, D_{AS}, D_{SA}, D_{AFR}, D_{AUS}) connected through network switches (sw). Clients, c, are served by their closest datacenter. The master datacenter hosts four tier components - application server tier (T_{app}), database server tier (T_{db}), file server tier (T_{fs}) and index server tier (T_{idx}) - while slave datacenter components only host T_{fs} . T_{fs} and T_{db}tiers are backed by their respective Storage Area Networks (san). Individual server components encapsulate low-level components representing hardware re-sources: memory (mem), processor (cpu), network card (nic) and disk arrays ($raid$). These components are modeled using queuing network models.

Operations define the interactions between clients distributed across continents and the software application running on the infrastructure. Examples of typical operations are LOGIN, SEARCH or OPEN. Each application consists of a set of client-initiated operations. Background jobs are also represented as operations, but these are periodically scheduled by the datacenter. Operations are decomposed into trees of messages that flow between datacenters, server tiers and hardware components affecting their internal status. Each message in the cascade represents an interaction between components in the system and encapsulates information about

the associated resource allocation and processing cost. This information is reflected by four hardware-agnostic parameters: Data transfer in KB (R_t), memory footprint in KB (R_m), processing cost in thousands of cycles (R_p) and disk read/write in KB (R_d). These parameters dictate the work to be carried out in queueing networks and/or links, and hence, determine the final duration of the operation. The cumulative processing time of the sequence of messages pertaining to the same operation constitutes the operation response time. Figure 8 illustrates the decomposition of a classic file opening (OPEN) operation into a sequence messages. First, the client $c$in EU makes a request to T_{app}in the master datacenter, $D_{NA}$for the token needed to download the latest version of a file from the T_{fs}inD_{EU}. The T_{app}checks for metadata about this file in $T_{db}$to make sure T_{fs}in D_{EU}has indeed the latest version, if not, a synchronization request between the T_{fs}in DNAand the T_{fs}in D_{EU}would be triggered through the network link $L_{EU\rightarrow NA}$. Upon token reception by c, the token is used to download the file from T_{fs}in D_{EU}. The total response time of the OPEN operation is calculated by adding the time measured at each step, as shown in equation 1.

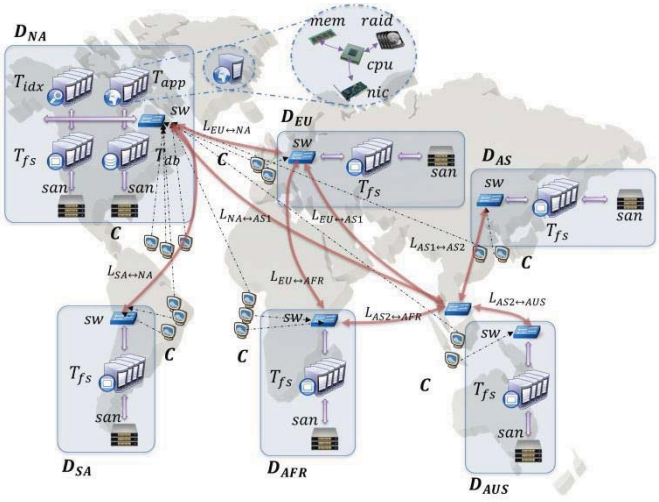

Figure 7: Global Data Infrastructure

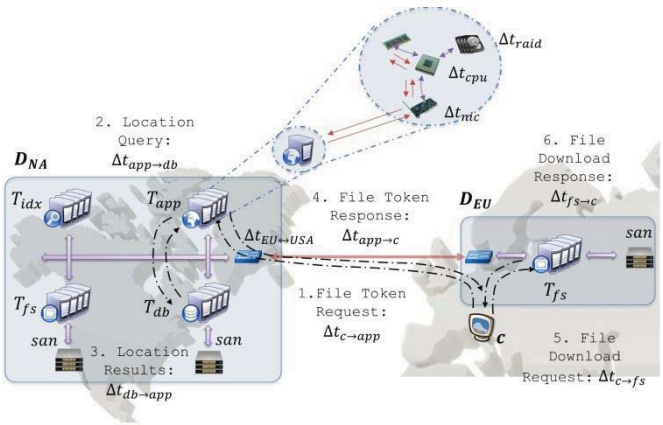

Figure 8: File Open operation decomposition

Equation 1: Calculation of OPEN duration

$$t_{open} = \Delta t_{c\rightarrow app} + \Delta t_{app\rightarrow db} + \Delta t_{db\rightarrow app} + \Delta t_{app\rightarrow c} + \Delta t_{c\rightarrow fs} + \Delta t_{fs\rightarrow c}$$

Each step is further decomposed into the messages exchanged by low-level components. For example, the token request from $c$to T_{app}:

Equation 2: Decomposition into low-level messages

$$\Delta t_{c\rightarrow app} = \Delta t_{EU\rightarrow NA} + \Delta t_{nic} + \Delta t_{cpu} + \Delta t_{raid}$$

The processing time of an individual message going through a low-level component depends on its state, which is affected by messages originated by other clients being processed simultaneously. The collection of components exchanging messages constitutes a Multi-Agent System (MAS) that reproduces the behavior of a global data infrastructure. Recently, MAS have been successfully utilized to simulate self-organizing computer networks [6], Service Oriented Architectures (SOA) [7] and High Performance Computing (HPC) systems [8]. To the best of our knowledge, this is the first time a MAS is used to simulate an IT infrastructure composed by several globally distributed multi-tier datacenters.

B. Simulator Validation

The main inconvenient that prevents medium size organizations from implementing global infrastructure-wide profiling mechanisms is the high cost and the incapacity to predict the behavior of the system when parameters are modified. Nevertheless, companies still maintain dedicated laboratories that test the side effects of new features, upgrades and configuration changes by profiling downscaled versions of their infrastructures in production against synthetic workloads. In this section, the reutilization of the profiling data collected in the laboratory towards the validation of the queuing network model of the simulator is proposed. The downscaled version of the infrastructure used for validation is composed by a single datacenter, D, composed by four tiers T_{app}^{1x6}, T_{db}^{1x6}, T_{fs}^{1x4}, T_{idx}^{1x4}, where T_{db} and T_{fs} are connected to two san^{1x20} storage networks through 4 Gbps connections. T^{axb} indicates that the tier is composed by a servers with b cores each. Similarly, san^{axb} indicates that the storage network is composed by $a$servers and with $b$disks each. Tiers are connected through a 1 Gbps network. Profiling is carried out over the execution of a classic CAD application composed by eight operations: LOGIN, TEXT-SEARCH, EXPLORE, SPATIAL-SEARCH, FILTER RESULTS, SELECT, OPEN and SAVE. Each operation will have three variants: light (minimum resource requirement), average (average resource requirement) and heavy (maximum resource requirement).

The (R_t, R_m, R_p, R_d) parameter set for each message for each operation variant is obtained by profiling the execution of each of these operations isolated within the laboratory infrastructure. A series is defined as a serial sequence of these 8 operations pertaining to the same variant family launched by the one client every $\overline{\Delta t}$. The synthetic workload is generated by initiating a light series, average series and heavy series every $\Delta t(0)$, $\Delta t(1)$, $\Delta t(2)$ seconds respectively. Three experimentswere carried out: $\overline{\Delta t} = \{10\ 12\ 15\}$, $\{12\ 29\ 48\}$, $\{15\ 36\ 60\}$. During these experiments multiple series overlap and messages pertaining to multiple operations flow concurrently through the queuing network model. In this paper, we focused on the results for the validation of CPU utilization. The results that compare the proposed model against the laboratory experiment are presented in Table 2. The minimum Mean Percentage Error (MPE) observed was 4.97% for the file server CPU and the maximum MPE was 11.23% for the database CPU.

Table 2: Mean Percentage Error

$\overline{\Delta t}$	T_{app}	T_{db}	T_{fs}	T_{idx}
10-24-40	9.52	11.07	5.66	7.13
12-29-48	9.44	11.23	4.97	6.75
15-36-60	8.67	9.87	5.72	7.42

VI. Coherent Shared Memory Architectures

Simulations incorporating large domains or databases may necessitate intensive computations that can become intractable using serial processing. The solution to this problem is to employ parallel computing, which has historically been undertaken using distributed-memory clusters. However, one of the hindrances to scalable, cross-machine distribution of numerical methods is the communication of data between non-coherent memory at synchronization times, which is plagued by the latency of TCP/IP communication. Additionally, software development in this context requires specialist programming skills in message passing (i.e. MPI).

The performance and implementation challenges of distributed memory parallelism can be overcome by utilizing coherent shared memory hardware. This can be realized in two ways, either via a hard-wired array of multicore processors or by aggregating traditional clusters, via a software layer, into a single virtual system [9]. The latter is represented schematically in Figure 9. Both approaches have relative strengths and weaknesses; however they both achieve the most important requirement for simplifying the development and optimizing the performance of parallel software, which is to allow the data associated with each distributed computational task to be directly addressable by all other tasks.

Shared memory multicore processors have emerged in the last five years as a relatively inexpensive "commercial-off-the-shelf" hardware option for technical computing. Their development has been motivated by the current clock-speed limitations that are hindering the advancement, at least in terms of pure performance, of single processors [10]. With the expense and high demand for compute time on cluster systems, multicore represents an attractive and accessible HPC alternative, but the known challenges of software development on such architectures (i.e. thread safety and memory bandwidth issues [11, 12, 13, 14]) must be addressed. Multicore technologies are importantly beginning to infiltrate all levels of computing, including within virtualized, coherent shared memory architectures. As such, the development of scalable programming strategies for multicore will have widespread benefit.

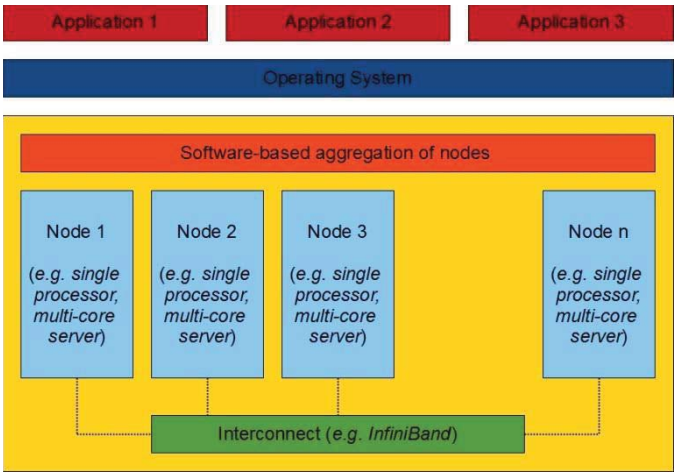

Figure 9: The aggregation of a traditional cluster, via a software layer into a single virtual system in which applications can directly address all memory

As already stated, one of the hindrances to scalable, cross-machine distribution of numerical methods is the communication of shared data (ghost regions). Depending on the model used, the communicated fraction of the problem domain can exceed 50%. In this situation Amdahl's Law [10], and the fact that traditional cross-machine parallelism using messaging packages is a serial process, dictates that this type of distributed memory approach will scale poorly.

If a problem is divided into spatial sub-domains for multicore distribution, ghost regions are no longer necessary because adjacent data is readily available in shared memory. Sub-domains can take on any simple shape or size and threaded programming means many small sub-domains can be processed on each core from an events queue rather than needing to approximate a single large, computationally balanced domain for each processor. Consequently, dynamic domain decomposition becomes unnecessary.

The decomposition of the spatial domain of a numerical method creates a number of computational tasks. Multicore distribution of these tasks requires the use of a coordination tool to manage them onto processing cores in a load balanced way. While such tasks could easily be distributed using a traditional approach like scatter-gather, here the H-Dispatch programming model of [15] has been used because of the demonstrated advantages for performance and memory efficiency. This library is based on the Concurrency and

Coordination Runtime (CCR) of Chrysanthakopoulos and co-workers [16, 17] and earlier work by Stewart [18].

The novel feature of H-Dispatch (see Figure 10) is the way in which tasks are distributed to threads. Rather than a scatter or push of tasks from the manager to threads (i.e. the OpenMPmodel [19]), here threads request values when free. H-Dispatch manages requests and distributes cells to the requesting threads accordingly. It is this pull mechanism that enables the use of a single thread per core as threads only request a value when free, thus, there is never more than one task at a time associated with a given enduring thread (and its associated local variable memory). Additionally, when all tasks in the problem space have been dispatched and processed, H-Dispatch identifies step completion (i.e. synchronization) and the process can begin again. Importantly, it was found that the H-Dispatch distribution model facilitated adjustable cache-blocking which allowed performance to be tuned via the computational task size.

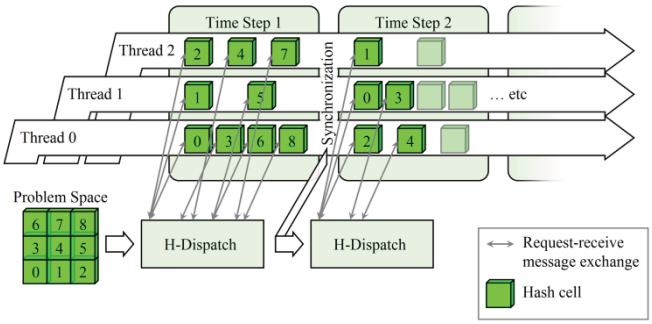

Figure 10: Schematic representation of the H-Dispatch programming model

Testing of this approach to multicore parallelism on a 24-core server found near-linear speed-up and peak efficiencies of 95% in fluid dynamics simulations. By varying the computational task size, the importance of optimal cache block size was also demonstrated.

VII. Conclusions

This paper illustrates the potential of adapting large coherent shared memory computing environments to the storage and processing of large data sets. We conclude that in-memory strategies can be applied to data sets on the order of tens of Terabytes and in the near future to Petabytes. The benefit of modeling future global systems "before" major architectural decisions are made is demonstrated. The elevation of security to being perhaps the most important design requirement will require the rethinking of many of our critical systems.

References

[1] Twitter, Inc., "Twitter api." [Online]. Available: http://apiwiki.twitter.com.

[2] frogdesign, "A world of tweets." [Online]. Available: http://aworldoftweets.frogdesign.com/

[3] "Nyc.com event calendar." [Online]. Available: http://www.nyc.com/events/

[4] K. Toutanova, D. Klein, C. D. Manning, and Y.Singer. "Feature-rich part-of-speech tagging with a cyclic dependency network," in Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, pp. 173-180, Morristown, NJ, USA: Association for Computational Linguistics, 2003.

[5] D. G. Hart. "Using AMI to Realize the Smart Grid," *IEEE*, 2008.

[6] M. Niazi and A. Hussain. "Agent-based tools for modeling and simulation of self-organization in peer-to-peer, ad hoc, and other complex networks," Comm. Mag., 47:166–173, March 2009.

[7] S. Karnouskos and M. M. J. Tariq. "An agent-based simulation of soa-ready devices," in Proceedings of the Tenth International Conference on Computer Modeling and Simulation, pp. 330–335, Washington, DC, USA: IEEE Computer Society, 2008,

[8] Y.-Q. Huang, H.-L. Li, X.-H.Xie, L. Qian, Z.-Y.Hao, F. Guo, and K. Zhang. "Archsim: a system-level parallel simulation platform for the architecture design of high performance computer," J. Comput. Sci. Technol., 24:901–912, September 2009.

[9] www.scalemp.com

[10] M. Herlihy and N. Shavit. The art of multiprocessor programming. Morgan Kaufman, 2008.

[11] K. Poulsen. "Software bug contributed to blackout," Security Focus, 2004.

[12] J. Dongarra. D. Gannon, G. Fox, and K. Kennedy. "The impact of multicore on computational science software," CTWatch Quarterly, 3(1), 2007.

[13] C. Leiserson and I. Mirman. "How to survive the multicore software revolution (or at least survive the hype)," Cilk Arts, Cambridge, 2008.

[14] N. Singer. "More chip cores can mean slower supercomputing, Sandia simulation shows", Sandia National Laboratories News Release, 2009. Available: http://www.sandia.gov/news/resources/releases/2009/multicore.html

[15] D. Holmes, J. Williams, and P. Tilke. "An events based algorithm for distributing concurrent tasks on multicore architectures," Comput. Phys. Commun., 181(2):341-354, 2010.

[16] G. Chrysanthakopoulos and S. Singh. "An asynchronous messaging library for C#," in Proceedings of the Workshop on Synchronization and Concurrency in Object-Oriented Languages, pp. 89-97, San Diego, 2005.

[17] X. Qiu, G. Fox, G. Chrysanthakopoulos, and H. Nielsen. "High performance multi-paradigm messaging runtime on multicore systems," Technical report, Indiana University, 2007. Available: http://grids.ucs.indiana.edu/ptliupages/publications/CCRApril16open

[18] D. Stewart, R. Volpe, and P. Khosla. "Design of dynamically reconfigurable real-time software using port-based objects," IEEE T. Software Eng., 23:759-776, 1997.

[19] M. Curtis-Maury, X. Ding, C. Antonopoulos, and D. Nikolopoulos. "An evaluation of OpenMP on current and emerging multithreaded/multicore processors," in M. Mueller, B. Chapman, B. de Supinski, A. Malony, and M. Voss, editors, OpenMP Shared Memory Parallel Programming, Lecture Notes in Computer Science, 4315/2008: pp. 133-144, Springer, Berlin/Heidelberg, 2008.