7-26-2005

The information age: challenges and opportunities for control and engineering

Chaouki T. Abdallah

J. D. Birdwell

J. N. Chiasson

M. M. Hayat

Z. Tang

See next page for additional authors

Recommended Citation

Abdallah, Chaouki T.; J. D. Birdwell; J. N. Chiasson; M. M. Hayat; Z. Tang; and J. White. "The information age: challenges and opportunities for control and engineering." *29th Annual International Computer Software and Applications Conference* 2005, 1, (2005): 77-82. doi:10.1109/COMPSAC.2005.96.

The Information Age: Challenges and Opportunities for Control and Engineering

J. D. Birdwell*, J. N. Chiasson*, C. T. Abdallah+, M. M. Hayat+, Z. Tang*, J. White*
*Dept. of Elec. and Comp. Engr.
University of Tennessee
Knoxville, TN 37996-2100 USA
{birdwell, tang, jwhite}@lit.net, chiasson@ece.utk.edu
+Dept. of Elec. and Comp. Engr.
University of New Mexico, EECE Building, MSC011100
Albuquerque, NM 87131-0001 USA
{chaouki, hayat}@eece.unm.edu

Abstract

The proliferation of computational resources and data networks has enabled the conception and solution of information management problems of an unprecedented scale. These technologies expose vulnerabilities that may be exploited through happenstance or design. The engineering problems encountered in the design of networked data systems and software are familiar in the controls community: resource allocation and management, distributed decision-making, time-delay systems, discrete-event and hybrid systems analysis and design, detection and identification, stability, robustness and reliability, and large-scale systems.

1. Introduction

This presentation offers a perspective, and perhaps some insight, into a few of the possible applications of systems and control technologies in information resource management and operation. The emphasis is upon the roles that systems dynamics, measurement, information structures, and controls processes play in determining achievable performance. Solutions typically require a combination of art and science or mathematics – they go beyond the current capabilities of formal design methods and analysis techniques. Fast processors and high-bandwidth communications networks provide the foundation for high-performance computing. The dynamics these systems exhibit include the effects of time delays, processing and bandwidth constraints, and random behaviors caused by both processor and network loadings and unmodeled or unknown decision processes within operating systems. The transient behavior of a novel load balancing method for a parallel database is discussed. Measurement of performance in a high-performance parallel computation environment is non-trivial, and leads to issues surrounding the synchronization of distributed clocks. Grid computing is parallel computation using a geographically distributed, and typically very large, collection of computers, with shared networking resources, which lead to increased and more variable delays in the transmission of both data and control information, and degradation of performance. Reliability of computational and network resources, and redundancy of information, are exceptionally important issues that lead to dynamic resource allocation strategies in the presence of delays and uncertainties. Related problems are encountered for high-bandwidth long distance communications channels. As our societies become dependent upon our information infrastructure, the security and reliability of that infrastructure is of paramount importance. Information security in the enterprise, or in distributed computational grids, requires dynamic responses to detected threats, which can originate from within or outside the protected infrastructure. A closed-loop control perspective is valuable, as the engineer designs instrumentation and controls to detect threats, identify threat patterns, and take actions to reduce vulnerabilities. Information surveillance generates massive collections of data, and automated knowledge discovery, or data mining, methods are required. Dynamic systems offer one solution strategy, whereby objects of a collection are associated with states of a nonlinear system whose fixed points represent naturally occurring clusters of similarity within the collection. The technology has wide applicability, with implications in digital libraries, counter-terrorism and homeland security, and analysis of molecular structures. Many

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 IEEE

of the flaws of today's software systems, which introduce security vulnerabilities, can be traced to the software engineering processes used to develop these systems. This can be understood from the perspective of systems dynamics and control. Methods of software engineering, such as the software lifecycle processes advocated by IEEE standards, are in essence coupled feedback loops, where development and lifecycle costs can be related to loop time delays and gains, and, in some instances, the absence of appropriate review and feedback. The fields related to software and control-ware are synergistic: Both can learn from the other. This presentation provides a glimpse into application areas where methods and insights from systems and control engineering can have immediate and positive impacts. The applications, in turn, stress the systems and control engineering field, reminding its practitioners of their limitations: Design methods for hybrid control strategies, especially for large scale or distributed systems, are still in their infancy. While substantial theory exists for analysis of systems that incorporate fixed time delays, much less is known where communications channels exist that packetize information, transmit data across band-limited channels, and subject the information packets to random delays. There are abundant opportunities for systems and control engineering practitioners to make significant contributions to society while addressing open and difficult fundamental problems in their chosen field.

2. Load Balancing

Many computational problems have outgrown the capacity of single processor machines. Parallel computing, or the use of multiple communicating processors to solve a single problem, originated in communities that required intensive numerical calculations, but parallelism is now found in a large variety of applications. Examples include large web services, such as those offered by Google, and require parallel designs for resource management, file systems, web servers, and databases. This research group has focused upon parallelization of specialized databases for storage of DNA profiles. Two important research areas have been *load balancing* and *dynamic resource allocation*. Load balancing methods attempt to level out the processing workload across a collection of computers, or nodes, in a parallel machine, while dynamic resource allocation addresses problems with contention among tasks for resources and reassignment of resources in the event of changes in the hardware architecture, through either failure or installation of components. In both cases, a control systems perspective has been adopted.

In the DNA database application, a database search engine is executed on each node of a parallel machine. New tasks enter the parallel system from clients and are inserted in task input queues attached to each search engine, typically selected randomly from the population. New tasks can also be created by the search engines as tasks are evaluated. The database is hierarchical, and processor nodes are partitioned into groups that contain identical data. Within each group, any task can be performed by any node, and it is advantageous to supervise the assignment of tasks to members of the group in order to equalize the waiting and processing times tasks entering the nodes experience. The parallelization is fine-grained, meaning that each task requires little time (typically less than 1 msec), and for this reason communication delays between nodes are important. The delay experienced by a packet of information moved between nodes depends upon its size, and there are substantial random effects caused by the operating system of each node and communications network loading. A flow model has been used to model these task processing and load balancing processes [1][12].

A continuous state task processing and load balancing model is advantageous in that dynamic properties such as stability, consistency, and task conservation can be analyzed, and precise results can be obtained. However, several of the assumptions required by this type of model are not realistic. For example, the continuous state model considers all tasks within each queue as homogeneous – this fails to capture effects that are a consequence of relative positions of tasks within the queue or each task's internal state. Second, data transmissions between nodes are encapsulated within packets. Each packet experiences its own delay, which varies from packet to packet, and the size of a packet influences this delay. Finally, load balancing actions are not performed continuously, but are instead performed on a scheduled basis in a discrete fashion. All of these characteristics of the actual system affect performance, and none are accurately captured by the continuous state model. To study these effects, a simulation model has been developed using OPNET Modeler. An example illustrating the impact of the placement of tasks on the queues is provided.

OPNET Modeler is a tool suite for the creation and analysis of discrete event simulations of computer networks. A network in OPNET Modeler is built of many components including network models, node models, link models, packet formats, and process models. A model was created to simulate the load balancing algorithm's behavior with time-varying delays over different network topologies. A comparison of results obtained with this model, relative to results from experiments conducted using a parallel computer, has been published [13].

In the OPNET model of a load balancing system, the nodes are connected by a model of a gigabit switch. Each node simulates the load balancing algorithm using a process model with a given set of initial conditions. The process model is a finite state machine, in which events from

the simulation kernel produce conditions that cause transitions between states. The time required for a packet to reach its destination is modeled in the simulation by two delays, a transfer delay and a propagation delay. The transfer delay is deterministic and proportional to the size of the exchanged data. The propagation delay is a random variable with a Poisson distribution plus an offset, which accounts for deterministic delays. The random component accounts for the variable time delays that come from network protocol stacks, network congestion, and packet fragmentation.

Additional tasks can be created when a task is processed. Wild card matches, and matches to within some threshold of an exact match, can cause this behavior in the DNA database developed by Birdwell *et al.* [4][2][3][5]. In these cases, a search through a tree structured index may require traversal of multiple branches from a given index node. The task generation model assumes that if a task spawns n new tasks, each spawned task will spawn $n-1$ tasks, and so on as the index tree is descended. A search can create additional tasks, leading to load imbalances.

To study this effect, a geometrically decreasing (in n) probability that each task will generate n additional tasks is specified. Each task is randomly assigned a child value in the set $[0, m]$, where $m > 0$ is the maximum number of jobs a task can spawn. The probability that a task has a child value of n is $p(n) = \beta^n/\alpha$, where $\alpha = (\beta^{m+1}-1)/(\beta-1)$. When a task is processed, if its child value is greater than 0, it spawns n new tasks that each have child values of $n-1$; these tasks will in turn each spawn $n-1$ tasks, and so on.

When new tasks are generated in a discrete event simulation, one must decide where to place them. This is substantially different from the continuous state flow-based load balancing model, which treats all tasks the same and models only the total number of tasks in each queue. New tasks can be placed either at the head (meaning they are next in line for execution) or the tail of the queue, and this decision has a large impact on system performance. To illustrate this, two experiments were performed in which one task is inserted into the queue of a node with a child value of 5. This causes 325 additional tasks to be generated. If the dynamically generated tasks are viewed as a tree, then processing each node of the tree generates the next level of tasks beneath that node. Inserting tasks to the tail of the queue is equivalent to processing this task tree in a breadth first fashion. The size of the queue will grow to a large value and then decrease, as shown in Fig. 1. Placing the tasks at the head of the queue is equivalent to processing the tree depth first. This spreads the task generation out over time, giving the processor time to catch up, as shown in Fig. 2. Both methods end at the same time, but their transient behavior is quite different.

While placing new tasks at the head of the queue may seem to make the transient behavior of the queue lengths

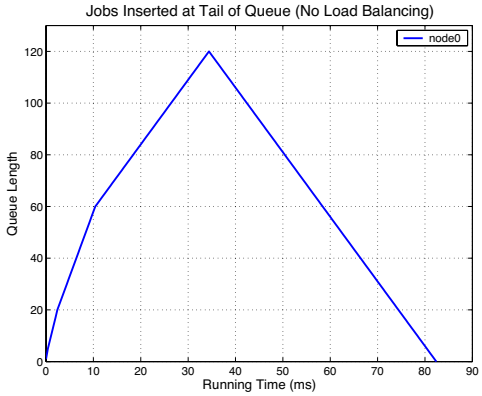

Figure 1. Queue size vs. time with jobs inserted at tail of queue (no load balancing).

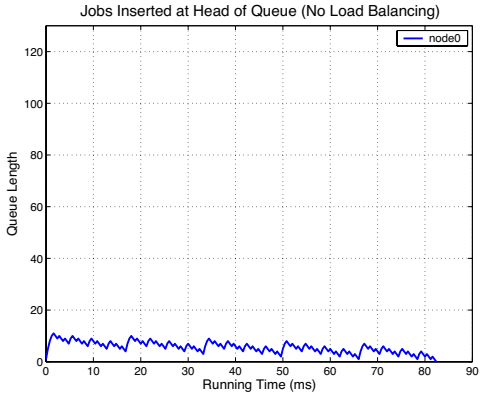

Figure 2. Queue size vs. time with jobs inserted at head of queue (no load balancing).

better, this actually results in poorer performance of the load balancing algorithm. The load balancer responds to the large explosion of new tasks and is able to distribute them across available resources, as shown in Fig. 3. The depth first approach draws out the task generation process, causing the node's queue length to only occasionally grow above the threshold at which load balancing is performed, as shown in Fig. 4. As a result, other nodes sit idle and the completion time increases.

A few conclusions can be drawn. First, the placement of newly generated tasks upon queues (at the beginning or end of the queue) impacts the evolution of queue size. This placement has a dramatic effect upon load balancing performance. Placement of these tasks at the head causes the system to preferentially service the current database search tasks to completion, causing lower service times. However, load balancing does not occur in an efficient manner un-

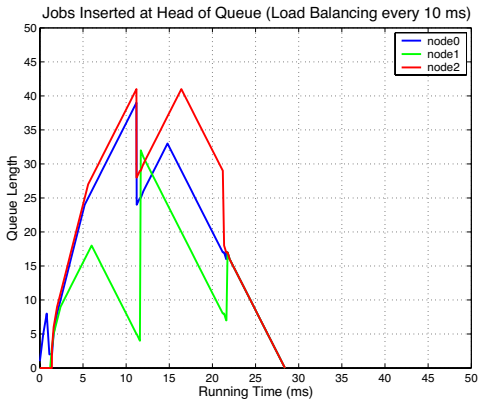

Figure 3. Queue size vs. time with jobs inserted at tail of queue (load balancing at 10 msec intervals).

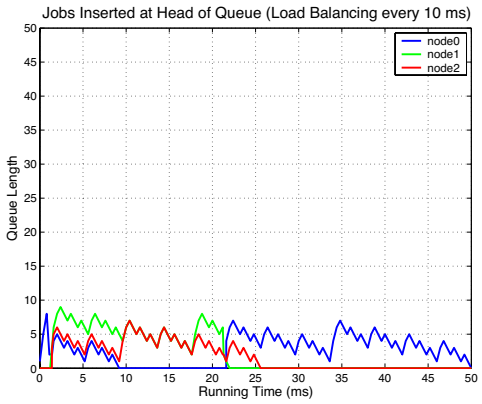

Figure 4. Queue size vs. time with jobs inserted at head of queue (load balancing at 10 msec intervals).

der this strategy, and some nodes of the parallel computer may remain idle, extending the time required to complete all tasks and lowering throughput (measured as average number of tasks processed per second). In contrast, placement of newly created tasks at the end of the queue causes larger queue sizes. Task service times can be adversely affected, since intervening tasks must be serviced first. But, load balancing can transfer tasks to other nodes, resulting in lower time to completion of all tasks and higher throughput.

3. Information Security

The information security field takes many forms: Three examples are protection against external, but uncoordinated threats such as computer viruses, detection of attack patterns, especially those that are coordinated from many hosts, and response in real-time to reduce their impacts, and prevention of the compromise of data and resources through misuse by internal personnel. Information security is a "hot" topic, in part, because software and hardware vendors have taken threats less than seriously in the past, and many vulnerabilities exist throughout computer operating systems and applications.

The design of information system resources that can respond dynamically to threats has similarities to the development of command, communications, and controls in the systems and controls field. An adversary's goals are to methodically attack distributed but communicating resources in a manner that overwhelms defenses, leaves additional portions of the networked systems more vulnerable to attack, and, when possible, co-opts resources for the adversary's use. Attacks and responses occur on extremely short time scales, which mandates automated responses. However, full automation is not advisable because the most damaging attacks are often the ones that have not been observed in the past.

An example information security system that incorporates a dynamic response is the dynamic firewall. Firewalls typically either allow all traffic except for specific types that are disallowed, or deny all traffic except types specifically allowed. Of these two approaches, the second carries less risk. Firewalls use rules to specify types of traffic that are allowed (or denied), and, if allowed, how the traffic is to be routed. These rules can be modified dynamically in response to detected events. For example, the Port Sentry tool, developed by Psionic Technologies [1], detects attempts to connect to or otherwise scan network ports on a computer connected to a network. If an attempt is detected, rules are used to perform actions, which can include modification of the firewall's rules to disable the originating host's access to the firewall. One example of this approach is documented in [14]. In a more distributed setting, firewall rules can be implemented by a network router, and these rules can be changed dynamically to disallow or redirect traffic classified as malicious.

Sophisticated attacks can originate simultaneously from multiple sources on a network; an example is the class of distributed denial of service attacks. A post-attack forensics analysis of collected data may help determine the origin of an attack. This is an example of "link analysis" in the law enforcement and military communities: Given many entities (computers, service providers, and individuals), and given relationships between entities, where the relationships are tagged by, for example, type and time, find clusters fitting a specified pattern of entities and relationships.

1 See http://www.securityfocus.com/infocus/1580.

Link analysis problems can be formulated as "link discovery," or "knowledge discovery" problems, whereby all clusters fitting a specified pattern are found from a large collection of data. These problems have broad applicability: digital libraries (clusters of similar publications or papers that cross reference each other within a collection), antiterrorism (communications with similar content, among a group of individuals, or with a causal relationship), bioinformatics (clusters of regions within a genome or of protein sequences that exhibit or express similar properties), and law enforcement (money laundering, illicit drug transactions, communications traffic analysis) are a few applications beyond the area of information security.

Birdwell developed an approach [7] to link discovery that embedded the entities of a database in three-dimensional Euclidean space and defined forces on their representations that caused them to cluster into groups of entities that are highly linked. For each particle of this multi-body system, attractive forces were defined for every particle with a relationship to the particle. Repulsive forces, similar to an "anti-gravity" force, were defined between every pair of particles. Simulation of this dynamical system caused those particles linked by their relationships to self-organize into clusters, while particles that were not related were forced apart. The dynamic system was defined so that an equilibrium was reached, resulting in a stable link diagram that could be used to navigate among the clusters and links.

4. Software Life Cycle Processes

A software life cycle model is defined by IEEE/EIA standard 12207 [10] as:

> A framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use.

Several frameworks exist, and large bodies of literature have evolved for each. A software life cycle process describes the processes used to define, develop, support, and retire software systems, from conception through death. The article by Cai [6] provides an excellent survey of software cybernetics and control of software lifecycle processes. One of the popular frameworks is the waterfall model, which uses several interacting process stages, with information flowing from one stage to the next. Typical stages include a Specifications process, development of Software Requirements, providing an overall design of the software using a Software Design Document, Detailed Design processes, an Integration Plan, a System Report, and Operations. Incorporated with these forward process phases are verification, development of test procedures, and validation. These processes map to software engineering organizations, and, through trial and error, methods have been developed that enable organizations to establish software life cycle and engineering processes that produce software with reasonable quality and consistency. Example procedures and guidance for organizations that wish to establish these processes are provided in the IEEE Standards literature [9][8], together with additional standards for the various components, which are referenced in both of these documents.

In 1994, Birdwell and B. C. Moore developed, for an industrial client, a formulation of software life cycle processes based upon the precepts of systems and control theory. This work provided a basis for comparison with implementations of software life cycle processes within supplier organizations so that deficiencies could be identified. The result was a signal flow graph that emphasized the highly interconnected nature of the life cycle processes and the feedback mechanisms inherent in the processes. the feedback structures among the software life cycle processes.

Software life cycle processes are not, in these authors' opinions, amenable to the precise mathematical analysis to which control theorists are accustomed. However, the perspectives of control theorists and practitioners can provide significant insights. The structures of software life cycle processes that are effective have some features in common, due in large part to two characteristics of the problem domain: human factors, and time delays. Humans have two fundamental limitations that severely compromise their ability to develop quality software: First, they are not able to hold within short-term memory more than a few items or concepts (where "few" is often interpreted as around seven). This leads to the necessity for breaking down complex software development and maintenance problems into small pieces, and the definition of interfaces between these pieces.

Second, humans have a highly nonlinear observational error rate: It is difficult for individuals to find errors in their own code, or to adequately test code when they have participated in its design and development. In the parlance of the systems community, measurement error rates are much higher when people examine objects with which they are intimately familiar – assumptions are based upon their expectations, and these assumptions are violated by the implementation. This explains the necessity of separation of software development from software verification, testing, and validation – an individual should not participate in both teams.

Finally, time delays limit the performance of a software life cycle organization, where performance can often be directly related to cost. An organization that tightly integrates design/development and verification/testing/validation teams, so that small software components are tested, and

test results are rapidly fed back into the development process, should on average outperform an organization with a more loosely coupled structure. This observation appears to be borne out in practice, and the conventional wisdom is that unit testing should occur as rapidly as possible, and on small units of code.

We leave this section with a final observation: Consistency is important. This is related to human factors. An illustrative example can be found in the C++ programming language [11], where it is possible to implement interfaces to class operators several different ways. Developers also have great flexibility on what is, and what is not, implemented for user-defined classes, and how the functionality is implemented. An organization that defines implementation standards, picking one way to perform each task rather than allowing each developer his or her choice, is likely to see reduced error rates, or, stated another way, less cost per line of developed software. If for no other reason, staff members who did not participate in the development processes will be able to read and understand the software more easily if they do not have to be familiar with several dialects.

5. Summary

This paper provides an overview of three areas within the intersection of computer engineering, information technologies, and systems and control engineering where concepts from the modeling, simulation, and control of dynamical systems can provide insight and benefit: load balancing, information security, and software development and life cycle processes. Substantial opportunities exist for future work; this paper only provides a highly simplified overview of some of the contributions that the areas of dynamical systems and control technologies have made.

6. Acknowledgments

J. D. Birdwell, J. N. Chiasson, Z. Tang, and J. White were partially supported by the National Science Foundation under grant ANI-0312611. C. T. Abdallah and M. M. Hayat were partially supported by the National Science Foundation under grant ANI-0312611. We are grateful to OPNET Technologies, Inc. for providing an academic license for their OPNET Modeler network simulation software. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U. S. Government.

References

[1] J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, and T. Wang. Dynamic time delay models for load balancing Part I: Deterministic models. In K. Gu and S.-I. Niculescu, editors, *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, volume 38, pages 355–370. Springer-Verlag, 2004.

[2] J. D. Birdwell, R. D. Horn, T.-W. Wang, P. Yadav, and S. Niezgoda. A hierarchical database design and search method for CODIS. In *Tenth International Symposium on Human Identification*, September 1999. Orlando, FL, USA.

[3] J. D. Birdwell, T.-W. Wang, R. D. Horn, M. Rader, P. Yadav, V. Chupryna, P. Dasgupta, D. J. Icove, and S. Niezgoda. CODIS matching algorithm for large databases. In *Fifth Annual CODIS Users Group Meeting*, November 1999. Washington, DC, USA.

[4] J. D. Birdwell, T.-W. Wang, R. D. Horn, P. Yadav, and D. J. Icove. Method of indexed storage and retrieval of multidimensional information, 2004. U.S. Patent 6,741,983.

[5] J. D. Birdwell, T.-W. Wang, and M. Rader. The university of tennessee's new search engine for CODIS. In *6th CODIS Users Conference*, February 2001. Arlington, VA, USA.

[6] K. Y. Cai, J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur. An overview of software cybernetics. In *Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practices*, pages 77–86, 2003.

[7] R. D. Horn and J. D. Birdwell. Link discovery tool. In *Proceedings of the ONDCP/CTAC 1997 International Symposium*, August 1997. Chicago, IL, USA.

[8] Institute for Electrical and Electronic Engineers. *IEEE Std. 1074.1–1995: IEEE Guide for Developing Software Life Cycle Processes*, September 1995.

[9] Institute for Electrical and Electronic Engineers. *IEEE Std. 1074–1997: IEEE Standard for Developing Software Life Cycle Processes*, December 1997.

[10] Institute for Electrical and Electronic Engineers. *IEEE/EIA Std. 12207.0–1996: (ISO/IEC 12207) Standard for Information Technology — Software Life Cycle Processes*, March 1998.

[11] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, Boston, MA USA, 3rd edition edition, 2000.

[12] Z. Tang, J. D. Birdwell, J. Chiasson, C. T. Abdallah, and M. M. Hayat. A time delay model for load balancing with processor resource constraints. In *Proceedings of the 2004 Conference on Decision and Control*, pages 4193–4198, December 2004. Paradise Island, Bahamas.

[13] Z. Tang, J. White, J. Chiasson, J. D. Birdwell, C. T. Abdallah, and M. M. Hayat. Closed-loop load balancing: Comparison of a discrete event simulation with experiments. In *Proceedings of the 2005 American Control Conference*, June 2005. Portland, OR.

[14] T. Verwoerd and R. Hunt. Intelligent firewalls — a new technique. Technical report, University of Canterbury, Christchurch, New Zealand, 2002. http://coscweb2.cosc.canterbury.ac.nz/research/RG/i-net_security/IF.pdf.