QoS Aware Dependable Distributed Stream Processing

Vana Kalogeraki[1]*, Dimitrios Gunopulos[1], Ravi Sandhu[2], Bhavani Thuraisingham[3]
[1]Department of Computer Science & Engineering, University of California Riverside
[2]Institute for Cyber Security, Univ. of Texas at San Antonio
[3]Department of Computer Science, University of Texas at Dallas
{vana, dg}@cs.ucr.edu, ravi.sandhu@utsa.edu, bxt043000@utdallas.edu

Abstract

In this paper we describe our approach for developing a QoS-aware, dependable execution environment for large-scale distributed stream processing applications. Distributed stream processing applications have strong timeliness and security demands. In particular, we address the following challenges: (1) propose a real-time dependable execution model by extending the component-based execution model with real-time and dependability properties, and (2) develop QoS-aware application composition and adaptation techniques that employ resource management strategies and security policies when discovering and selecting application components. Our approach enables us to develop a distributed stream processing environment that is predictable, secure, flexible and adaptable.

1 Introduction

There is an increasing number of emerging stream processing applications executed on large-scale, distributed and heterogeneous computing systems. These applications will take advantage of computers and networks to enable geographically distributed groups of users to collaborate with one another, share data stored at different locations, share computing resources to execute computer-intensive experiments and visualize the experimental results in real-time. Such an infrastructure, coupled with advances in computational methods and software technologies, has stimulated the interest of users in multiple application domains. Examples of such domains include healthcare, surveillance, industrial process control, and financial applications. Unfortunately, unlike current platforms, future execution environments will be highly heterogeneous, constantly changing and of great complexity.

In this paper we focus on applications where the data are collected or delivered as streams, being either the output of different sensors (including optical, such as video, feeds), or provided as streams from content providers, such as servers that provide access to multimedia objects. This is particularly the case in health-care or surveillance applications, where data are collected by sensors, as well as in financial applications where it is important to follow indices over time. We also focus on fundamentally distributed applications where the data, and the information extracted from the data must be made available to a set of users located in geographically diverse positions. Such sensitive applications in emergency response, or in military and commercial environments, have strong timeliness and security requirements.

Efficient management of such a huge and widely distributed system is a considerable challenge. In our approach we have identified the following challenges that we have to address to develop dependable execution environments for the future large distributed applications. First, applications have end-to-end Quality of Service (QoS) demands, including predictable latency, jitter and reliability that must be satisfied simultaneously. The fundamental problem is that in a large-scale environment it is difficult to determine the resource usage of a specific application on a specific execution platform in advance and predict its resource requirements. Current distributed object and Grid middleware technologies provide standard protocols and platform-independent technologies that enable applications to interoperate and share heterogeneous resources across multiple administrative domains and computing platforms. However, (a) the distributed execution of the applications across multiple shared nodes, (b) the unpredictability in the arrival patterns of the requests and, (c) the multiple QoS requirements that need to be satisfied simultaneously make the prediction more complicated.

*This research was supported by NSF Award 0627191.

Second, resource changes and failures in a large-scale system are inevitable. Processors, networks and applications frequently fail or restart, service downtimes and routine maintenance are common processes. Predicting in advance and handling all these changes, significantly increases the complexity and difficulty of programming reliable applications. Our goal is to consider such a highly variable execution environment with significant changes in the application behavior and the availability of resources. Third, we need to ensure that the system as well as applications are secure, meet real-time constraints, ensure data integrity and support fault tolerant processing. This requirement is particularly important for critical stream processing applications. Finally, it is important for the resource management mechanisms to be transparent and non-intrusive so that can be easily employed by the user and thereby make them accessible to a much wider population of application developers and scientists.

2 Component based Execution Model

In this section we describe our component based execution model. Our objective is to build an environment where the user can easily describe the application, and the real-time and dependability requirements, and then the system can automatically modify the application and extend the composition entities to encode this functionality and define and enforce real-time and dependability policies on components during the execution which are essential to satisfy the user requirements.

To do that we need to develop an engine that takes as input the application graph (that describes the application execution) and the user requirements for QoS and dependability and creates a new, annotated, graph that satisfies the user requirements. Let G be the original application graph, R the user requirements, and let M (G, R) be the new graph. The requirements for QoS specify parameters such as deadlines and priorities, are given as a set of attributes. The security requirements are given, following the UCON framework, as lists of allowed accesses for each object. The process that will create M (G, R) must have the following information: generic descriptions of the components used by the application graph G, as well as a mechanism to insert the appropriate hooks in the components so that the required calls to UCON can be performed automatically.

2.1 Component based Model

Composing software from reusable building blocks (*i.e.,* components) has been a central idea in systems engineering. A *processing service component C* (or *component*) is defined as a processing element that implements an atomic processing function F_c on a set of input data streams $\sum is$ and produces a set of output data streams $\sum os$: $C = (F_c, \sum is, \sum os)$. Examples of processing functions include aggregation, filtering and correlation operations. Multiple instances of the same component can exist in the system. We denote these components as $c_1, .., c_k$. Components receive application data streams as inputs generated from external sources (or as the result of the execution of other applications) which are then processed by the components in real-time to generate outputs. Components can be composed dynamically to build complex distributed applications. An application is represented with an application component graph, a directed acyclic graph, where the vertices of the graph represent the components being invoked and the edges represent invocations between the components. Each component in the graph can be invoked in sequence or in parallel by multiple applications concurrently; thus, efficient sharing of components and streams is desirable, to improve the performance and scalability of the system.

2.2 Security Model

Recently Sandhu et al developed a new model called UCON for usage to unify all these extensions to traditional access control in a coherent, consistent and complete manner. We are extending the UCON model to develop RT-UCON or real-time UCON for application in the QoS aware stream processing space. Note that we have proposed some extensions to integrate Kane Kim's TMO (Time Triggers Message Triggers Object) model with UCON and presented our results in prior ISORC conferences. We are now examining these extensions for our distributed stream-based execution model.

In the UCON model[37], an access is not a permanent decision, but a process lasting for some duration with related actions. Besides, actions during an access period result in changes to subjects or objects attributes. Usage control model has the following components: subject, object, rights, condition and obligations. The authorization which consists of subject, object and access rights is the basic to access control as in traditional model. The UCON extends the general model of access control by including obligation and condition. The obligations are actions that are performed by subject or user before obtaining access rights and conditions are system or environmental restrictions such as system clock, system location, and system mode and so on. The distinguishing aspects of UCON compared to traditional models are continuity of access decision and mutability of subject and object attributes. There are three phases to complete access decision: before usage, ongoing usage and after usage. In UCON, access decision can be checked repeatedly during the access and revoked if some conditions are not satisfied or changed. The mutability and continuity of attributes make the UCON model very powerful and provide seamless security administration policy. Also, UCON model is useful to specify dynamic constraints and consumable access increasing or decreasing access time. Access decision is not a single function for subjects or objects, it interact with each other depending on access from others.

The UCON model encompasses the traditional model (e.g., MAC/DAC) and RBAC[37]. Security level of MAC and DAC and role of RBAC is adapted in UCON using attributes which are attached to subjects and objects. The basic authorization components of UCON and its mutability and continuity can be easily adapted for access control to the execution model. Mutability aspect of UCON can be used to control the scope of access by increasing or decreasing mutable attributes such as access time, access range and access count. The condition aspect of UCON also can be useful for the execution model by means of restricting the access. The component object can give permission or be denied during a particular time period, which is an extension from the idea of operations in the RBAC model. This feature may be used within the methods to restrict its behaviors or access from others. For example, methods can block the access from outside by changing its attributes or announce to all other objects when the system goes into maintenance or detect any abnormal status. Condition also can control resource usage to limit the number of access or scope. Through the entire system life or specific time period, conditions can be varied and security policies can be accommodated flexibly and dynamically to the execution scheme. This aspect of UCON is very beneficial and efficient for the execution scheme when its correlated with mutability. The concept of obligations also can give a useful way for access decisions. Obligations are active actions required to be performed on the subject side before accessing the object, so it can be used to screen any object before the subject acquires the access permission. Global timing synchronization can be one example of obligation. To communicate with each other, all objects must have the same global time to prevent abnormal behavior among the objects or to guarantee timing assurance.

The attributes of an object also can be controlled or referenced by obligation aspect as well. If an object does not allow using its resources associated with the attributes by other objects with respect to obligation, then access may be denied because it may allude to malicious objects. To accomplish the above features, object classes need to be extended to add attributes and methods for conditions, obligations and mutability of the UCON model. We may need to devise various methods to handle the components of the UCON model for real-time applications.

3 Developing a QoS aware dependable processing environment

Our plan is to develop *a QoS aware dependable stream processing environment* for enabling the composition and deployment of large-scale distributed applications. The following key challenges must be addressed:

- Develop a real-time dependable execution model by extending the component-based execution model with real-time and dependability properties.
- Develop QoS aware component composition policies, investigating component discovery techniques, resource management strategies and security policies.

3.1 Dependable processing for the execution model

We are now examining our extensions for our distributed stream-based execution model. We plan to investigate two aspects of dependable processing, real-time and access control properties.

Real-time Properties. The real-time interface is developed to ensure that the QoS requirements of the applications are met. To meet real-time requirements we need to specify parameters such as deadlines, execution times, priorities for the applications and fine tune parameters such as resource (CPU, memory, bandwidth) sharing, thread pools and buffer sizes for the execution environment. Our objective is to specify real-time and QoS parameters for the components offline and deal with the difficult problem of managing their execution at run-time to enforce the timing constraints to meet user requirements.

At application deployment time, a description of the components is provided, enriched with the real-time properties. An example of a real-time property is the application deadline. At run-time, the system reads the real-time properties of the components and assigns priorities to threads that will carry the execution of the components, as invoked by the application and based on the composition policy. As the application invokes components across multiple processors, the real-time property (*i.e.,* deadline) of the application is carried along with the invocations, along with other information about the application. This allows us to ensure that the real-time property is applied to the application end-to-end. As the application executes, the composition policies guarantee that the execution is based on the deadline of the application. We will first target single-threaded component implementations (in which each component is a single unit of control) and later address the more difficult problem of multi-threaded component implementations.

Component policies are used to implement composition. The component policy will define the name of the policy and the components for which the policy applies. Policies essentially define how components interact with their run-time environment. This separation effectively decouples a component from its execution environment so that the component can be deployed in different execution environments without modification to the component. The components are managed by the system. We employ profiling techniques to monitor the execution times of the components and local management techniques for CPU, memory, network, storage and other local resources [18] to schedule, configure and monitor the component execution on the local resources.

3.1.1 Access Control

For an execution model to be secure we need the secure composition of many components including secure real-time (RT) operating systems, networks, databases, middleware, and applications. We will investigate issues on enforcing access control policies for our execution model with applications running on various nodes cooperating to support a network of applications. We are taking advantage of some prior research results by our team members. Some early efforts on developing secure RT models at the MITRE Corporation [78]. We are continuing with this effort at UT Dallas [26]. A few efforts have been reported on incorporating security into RT middleware, especially RT object request brokers [41]. Designing flexible security in RT data management systems is reported in [38]. Access control policies specify the types of access that subjects have to objects. These policies include role-based policies where access to objects is granted depending on the roles of the users and usage control policies [3]. Our team member Ravi Sandhu from GMU is the inventor of two the prominent access control models: Role-based access control and Usage control (UCON) models [38]. He has demonstrated that UCON encompasses all the other models to date [26]. The challenge now is to incorporate UCON based access control into our execution model. For example, what credentials should client possess to obtain services of read-only or read-write types from servers? How can timing constraints and deadlines be met when the access control policies are enforced ? How can access control be enforced across a dynamically formed client-server chains across multiple nodes?

Access control for our execution model can also be viewed as the function of trust management and negotiation. Honoring a service request from a client say C1, depends on how much the server say S, trusts C1. What is the guarantee here that C1 will invoke client C3 and share the data obtained from client 2 only after informing C2? Recently there have been efforts on implementing trust management and negotiation systems for various types of information systems [11, 12]. UCON encompasses trust models, privacy models, dissemination models as well as access control. Therefore investigating UCON for our execution model will address the various issues that are important to develop a dependable programming environment.

A dependable execution model must not only satisfy the security policies but also ensure that the timing constraints are met. Enforcing security policies may be time consuming and as a result the system may possibly miss the deadlines. For example, in the military scenario that we described in section 2, the system may miss the deadline if several access control checks are performed. We need to develop flexible policies. Therefore in order to get the information to the war fighter as quickly as possible, security policies may have to be bypassed and later logged. This way the administrators will know that between time T1 and T2, the policies were bypassed. As stated earlier, we have examined UCON extensions for the TMO model. We are examining such extensions for the execution model.

3.2 QoS-aware Component Composition

To meet end-to-end real-time requirements for the applications, it is important to implement composition based on current resource availability. Our approach includes the following steps: (a) Component Discovery, and (b) QoS aware composition.

Component Discovery. There has been a large body of work on resource and data discovery in large-scale systems. However, the requirements of our QoS-aware composition are different from those of a traditional data sharing application: First, stream processing applications have QoS demands. To select the best components that meet the application QoS demands, we must consider multiple possible candidates along with the loads on the corresponding nodes. Second, our objective is to implement scalable dissemination techniques and are able to cope with frequent configuration updates.

Our approach is to employ data summarization and dissemination to propagate component and resource information to the appropriate nodes, using a fully decentralized and scalable mechanism. We propose to develop data summarization techniques based on the Bloom filter data structure[7]. Bloom filter is a compact way of representing summaries of data items stored at the nodes and testing to see whether a given item is included in the summary. The advantage of Bloom filters is that they provide a tradeoff between memory and traffic requirements and false positive ratio (i.e., false hits). Our system exploits the fact that a small number of false positives does not greatly affect the performance of our searching mechanism. This fact makes the Bloom filter approach highly suitable for locating components and streams accurately and fast. Furthermore, our system considers current processing and network loads when composing the applications. For example, if a component is available at two nodes, we select the component from the least loaded node.

QoS aware composition. For each application request, the user could specify the data source, application graph (that describes the application components and their corresponding invocations) and real-time requirements. Given the component and stream specifications, the node first queries the Bloom filters to get available stream and component information. As mentioned earlier, the Bloom filter can answer the query whether a specific component is available at the horizon of a node. If such a component exists, the system includes the component as a candidate component for the required function. The node then retrieves the load condition of each candidate component and the bandwidth in the communication links. The goal of our system is to reuse previously deployed components. We will investigate solutions in which we estimate the effect of component reuse by projecting the impact of the additional workload on the QoS of the affected applications and ensure that a component reuse won't degrade unacceptably the quality of the existing streaming applications[28]. Such a projection can assure the QoS provisioning for both current applications and the new application admitted in the system.

3.2.1 Adaptive Techniques for Flexible Dependable Execution

The assumption here is that in a distributed system any security mechanism can be compromised and the goal is to increase the probability the system will continue to operate in an uncompromised fashion despite the attacks. We will investigate tradeoffs with respect to the frequency and timing for the access control checks. Clearly the checks can be carried out at component invocation and response. This is the safest approach, but it is more efficient to do it once at application deployment time, if this can be done. An intermediary solution is to do the checks occasionally, but not in every invocation.

Consider the following situation: we have a long-lived stream processing operation, where a certain component is repeatedly invoked. Clearly we do not have to recheck the access control rights every time unless we expect them to change frequently. Consider the case where adaptation is based on changing availability of system resources. Here we can formalize the problem essentially in the same way that we formalize the composition problem, that is, if different levels of security can be defined we can decide which level to use based on user requirements and availability of resources. Here we can possibly consider optimizing two objective functions, (QoS and security) and either combine them into one using a user defined formula (for example a linear combination of the two objective functions) or optimize them separately.

Adaptation can also be carried out on changing user requirements. This also assumes that different levels of security can be defined. One idea here, as described in [30] and [42], is that processes operate at different access levels or with different credentials. Suppose process P1 cannot send any messages to Process P2 as specified by the policy. The security monitor will ensure that the policy rule is enforced. Furthermore, in such a situation, P1 will not be able to write into a stream object O that can also be read by P2. This is because if P1 can write into O and P2 can read from O, then P1 can pass information to P2 through O. However P1 could manipulate the read locks on O and covertly send data to P2 by acquiring and releasing read locks on O at different times. This is an example of a covert channel. So the security solution is to deny P1 read access whenever P2 has a write lock on O. But if P1 is a time critical process then it will need all the resources to complete its execution. In such a situation we need rules to resolve the conflicts. These rules may be specified by the users depending on the requirements. If the data in object O has to be read by P1 so that it has to be sent to the war fighter within 5 seconds, then P2 has to wait before it gets write access to O. The fact that there may be a potential for a covert channel has to be flagged so that the system does not let this situation repeat in order to limit the bandwidth of the channel.

We need to examine such adaptive policies for our execution model. The policies need to be flexible to handle the different user requirements. The goal is to enforce policies in such a way that for time critical applications the system must handle malicious attacks in such a way that the timing constraints are met. During non crisis situations such as during peace time, it is important that the system ensures that malicious attacks are prevented as much as possible by denying resources to malicious higher level processes.

4 Related Work

A component-based system consists of modules that implement a given interface, standardized by an external framework [25, 22, 23]. However, integration of real-time or dependability functionality is still an open research field. Work done so far includes efforts to specify real-time CORBA [13, 32], doctrines for distributed scheduling [24] and toolkits that provide dependency checks to support distributed embedded system development via components [35].

Distributed stream processing [1, 9] has been the focus of many recent research efforts.Load shedding techniques to trade-off processing precision with timely response are discussed in [36]. [20] describes an architecture for distributed stream management that makes use of in-network data aggregation to distribute the processing and reduce the communication overhead. Coherency requirements are used in organizing a hierarchy of data repositories to efficiently filter and disseminate streaming data in [33]. The problem of object or data discovery has been studied in the field of peer-to-peer overlays and data summarization and rumor spreading algorithms that offer probabilistic guarantees have been proposed [16, 12, 29].

Several efforts have attacked the problem of service composition in the context of multimedia streaming. SpiderNet [14] uses a probing protocol to setup the service graph, while in PROMISE [15] a receiver selects senders based on characteristics such as the offered rate, the availability, the available bandwidth, and the loss rate. In [10] the streams are transcoded before they are delivered to the receiver and quality adaptation is utilized to address changing resource availability. In this paper we focus on the dependable processing of streams in addition to delivery with QoS guarantees and extend the component based model by satisfying both real-time and dependability requirements.

Other real-time object models include the TMO model by Kan Kim et al [19], the AWACS models by Bensley and the RTSORAC model by Prichard, Wolfe et al [27].

Early security model include the discretionary access control models (DAC) and mandatory access control models (MAC) [2], [21]. In the 1990s there were many activities on developing different kinds of access control models. Notable are Sandhu's models on RBAC [31] and UCON [26]. In addition, security models for the semantic web [39], XML [5], [4] and RDF [8] have been developed. More recently there has been work on Trust, Privacy and Dissemination models [37], [6].

Integrating security and real-time systems research include the work reported by Thuraisingham Son et al, [42], [34] and the work on dependable semantic web and real-time dependable data mining by Thuraisingham et al [40] and the Secure TMO research being carried out at the University of Texas at Dallas by Jung-in and Thuraisingham [17].

5 Conclusions

In this paper we have proposed a QoS-aware dependable stream processing environment to deal with the challenges of real-time and secure processing of data streams in large-scale systems. It is our imperative design objective to develop an environment where the users can easily describe the stream processing application and define its real-time and dependability requirements, and then the system can extend the composition entities to encode this functionality and enforce real-time and dependability policies on components executing on the system resources to satisfy the user requirements. Our approach aims to provide a flexible and adaptable dependable execution environment.

References

[1] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *Proceedings of CIDR, Asilomar, CA*, January 2005.

[2] D. Bell and L. Lapadula. Secure computer systems : Unified exposition and multics interpretation. *Technical report ESD-TR-75-306 MITRE Corporation*, 1975.

[3] E. Bensley, P. Krupp, M. Squadrito, and B. Thuraisingham. Design of an object-oriented data manager and infrastructure for real-tine command and control system. *Proceedings IEEE WORDS*, 1996.

[4] E. Bertino, B. Carminati, E. Ferrari, and B. M. Thuraisingham. Selective and authentic third-party distribution of xml documents. *IEEE Transactions on Knowledge and Data Engineering*, 16:1263–1278, 2004.

[5] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Protection and administration of xml data sources. *IEEE Transactions on Knowledge and Data Engineering*, 43(3):237–260, 2002.

[6] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.

[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[8] B. Carminati, E. Ferrari, and B. M. Thuraisingham. Using rdf for policy specification and enforcement. *Proceedings DEXA Workshops*, pages 163–167, 2004.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of CIDR, Asilomar, CA*, January 2003.

[10] F. Chen, T.Repantis, and V. Kalogeraki. Coordinated media streaming and transcoding in peer-to-peer systems. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium, IPDPS, Denver, CO*, April 2005.

[11] F. Cuenca-Acuna and T. Nguyen. Self-managing federated services. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, SRDS, Florianopolis, Brazil*, October 2004.

[12] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable, replicated peer-to-peer systems. In *Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems, ICDCS, Providence, RI, USA*, May 2003.

[13] O. M. Group. Realtime corba joint revised submission, Mar. 1999.

[14] X. Gu and K. Nahrstedt. Distributed multimedia service composition with statistical QoS assurances. *IEEE Transactions on Multimedia*, 8(1):141–151, 2006.

[15] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using collectcast. In *Proceedings of the 11th ACM International Conference on Multimedia, Berkeley, CA, USA*, November 2003.

[16] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of ICDCS, Tokyo, Japan*, March 2004.

[17] K. Jungin and B. Thuraisingham. Secure tmo. *IEEE ISORC*, April 2006.

[18] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser. Dynamic scheduling for soft real-time distributed object systems. In *Proceedings of the IEEE Third International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 114–121, Newport, CA, March 2000.

[19] K. H. K. Kim, Y. Li, S. Liu, M. Kim, and D. Kim. Rmmc programming model and support execution engine in the tmo programming scheme. *Proceedings IEEE ISORC*, pages 34–43, 2005.

[20] V. Kumar, B. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-aware distributed stream management using dynamic overlays. In *Proceedings of ICDCS, Columbus, OH*, June 2005.

[21] C. E. Landwehr. Formal models for computer security. *ACM Comput. Surv.*, 13(3):247–278, 1981.

[22] Microsoft. Com: Delivering on the promises of component technology.

[23] S. Microsystems. Enterprise javabeans technology.

[24] Object Management Group. Dynamic Scheduling. Revised Submission, orbos/00-08-12, August 2000.

[25] Object Management Group. The Common Object Request Broker: Architecture and Specification. Edition 2.4, formal/00-10-01, October 2000.

[26] J. Park and R. S. Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[27] J. J. Prichard, L. C. DiPippo, J. Peckham, and V. F. Wolfe. Rtsorac: A real-time object-oriented database model. *DEXA*, pages 601–610, 1994.

[28] T. Repantis, X. Gu, and V. Kalogeraki. Synergy: Sharing-aware component composition for distributed stream processing systems. In *Proceedings of the CM/IFIP/USENIX 7th International Middleware Conference (Middleware 2006), Melbourne, Australia*, Nov-Dec 2006.

[29] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proceedings of IEEE INFOCOM 2002, New York, NY, USA*, June 2002.

[30] R. D. S. Son and B. Thuraisingham. An adaptive policy for improved timeliness in secure database systems. *Proceedings of the IFIP Database Security Conference*, pages 199–214, 1995.

[31] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[32] D. C. Schmidt and F. Kuhns. An overview of the real-time corba specification. *Computer*, 33(6):56–63, 2000.

[33] S. Shah, K. Ramamritham, and C. Ravishankar. Client assignment in content dissemination networks for dynamic data. In *Proceedings of VLDB, Trondheim, Norway*, September 2005.

[34] S. H. Son, R. Mukkamala, and R. David. Integrating security and real-time requirements using covert channel capacity. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):865–879, 2000.

[35] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. Vest: An aspect-based composition tool for real-time systems. In *IEEE RTAS*, pages 58–69, 2003.

[36] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proc. of VLDB, Berlin, Germany*, Sept. 2003.

[37] R. K. Thomas and R. S. Sandhu. Towards a multi-dimensional characterization of dissemination control. *Proceedings POLICY*, 2004.

[38] B. Thuraisingham. Multilevel security for distributed database systems - ii. *Computers and Security*, 10, December 1991.

[39] B. Thuraisingham. Security standards for the semantic web. *Computer Standards and Interface*, March 2005.

[40] B. Thuraisingham, C. Clifton, J. Maurer, and M. Ceruti. Real-time dependable data mining. *Proceedings of IEEE ISORC*, May 2005.

[41] B. M. Thuraisingham. Mandatory security in object-oriented database systems. *ACM OOPSLA*, pages 203–210, 1989.

[42] B. M. Thuraisingham and J. A. Maurer. Information survivability for evolvable and adaptable real-time command and control systems. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):228–238, 1999.