REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
DECEMBER 2008	Journal Article Postprint	April 2008 – December 2008

4. TITLE AND SUBTITLE

ADAPTIVE VOTING ALGORITHMS FOR A RELIABLE DISSEMINATION OF DATA IN FAULT-PRONE DISTRIBUTED ENVIRONMENTS

5a. CONTRACT NUMBER
In-House

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Kaliappa Ravindran, Kevin Kwiat, and Patrick Hurley

5d. PROJECT NUMBER
4519

5e. TASK NUMBER
22

5f. WORK UNIT NUMBER
49

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFRL/RIGA
525 Brooks Road
Rome NY 13441-4505

City University of New York
Department of Computer Science
New York, NY 10031

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
N/A

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TP-2009-16

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited PA#: WPAFB-2008-5054 Date Cleared: 14-August-2008

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Data collection in a distributed embedded system requires dealing with failures: data corruptions by malicious devices and arbitrary message delay/loss in the network. Replication of data collection devices deals with such failures by voting among the replica devices to move a correct data to the end-user. Here, a data voted upon can be large-sized and/or take a long time to be compiled. The goal of this paper is to engineer the voting protocols for good performance while meeting the reliability requirements of data delivery in a high assurance setting. Two metric quantify the effectiveness of voting protocols: Data Transfer Efficiency (DTE) and Time-to-Complete (TTC) data delivery. DTE captures the network bandwidth wasted and/or the energy drain in wireless-connected devices; whereas, TTC captures the degradation in user-level Quality of Service (QoS) due to delayed/missed data deliveries. Given the distributed nature of voting, the protocol-level optimizations to improve DTE and TTC reduce the movement of user-level data over the network, the number of control messages generated, and the latency in effecting a data delivery. The paper describes these optimizations, and reports experimental results from a prototype voting system.

15. SUBJECT TERMS
Device Replication, Malicious Faults, Device Heterogeneity; QoS and Performance of Data Delivery, Software Prototype; Fault-Tolerant Web Service

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Kevin A. Kwiat
U	U	U	UU	29	19b. TELEPHONE NUMBER (*Include area code*) 315-330-1692

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Adaptive voting algorithms for the reliable dissemination of data in fault-prone distributed environments

Kaliappa Ravindran*

Department of Computer Science
City University of New York (City College)
160 Convent Avenue
New York, NY 10031, USA
Fax: (212) 650–6248
E-mail: ravi@cs.ccny.cuny.edu
*Corresponding author

Kevin A. Kwiat and Patrick Hurley

Information Directorate
Air Force Research Laboratory
525, Electronic Parkway
Rome, NY 13441, USA
Fax: (315) 330–3875
E-mail: kwiatk@rl.af.mil
E-mail: hurleyp@rl.af.mil

Abstract: Data collection in a distributed embedded system requires dealing with failures: data corruptions by malicious devices and arbitrary message delays/loss in the network. Replication of data collection devices deals with such failures by voting among the replica devices to move a correct data to the end-user. Here, a data voted upon can be large-sized and/or take a long time to be compiled (*e.g.,* terrain surveillance images). The goal of our paper is to engineer the voting protocols for good performance while meeting the reliability requirements of data delivery in a high assurance setting. Two metrics quantify the effectiveness of voting protocols: Data Transfer Efficiency (DTE) and Time-To-Complete (TTC) data delivery. DTE captures the network bandwidth wasted and/or the energy drain in wireless-connected devices; whereas, TTC captures the degradation in user-level Quality of Service (QoS) due to delayed/missed data deliveries. Given the distributed nature of voting, our protocol-level optimisations to improve DTE and TTC reduce the movement of user-level data over network, the number of control messages generated, and the latency in effecting a data delivery. The paper describes these optimisations, and reports experimental results from a prototype voting system. The paper also describes a case study of voting deployed in web service access to information repositories.

Keywords: device replication; malicious faults; device heterogeneity; QoS and performance of data delivery; software prototype; fault-tolerant web service.

Reference to this paper should be made as follows: Ravindran, K., Kwiat, K.A. and Hurley, P. (2008) 'Adaptive voting algorithms for the reliable dissemination of data in fault-prone distributed environments', *Int. J. Business Intelligence and Data Mining*, Vol. 3, No. 3, pp.277–304.

Biographical notes: Kaliappa Ravindran is a Professor of Computer Science at the City University of New York, USA. Earlier, he had held faculty positions at the Kansas State University and the Indian Institute of Science and had also worked as a Control Systems Engineer at the Indian Space Research Organization. He received his PhD in Computer Science from the University of British Columbia, Canada. His research interests are in the service-level management of distributed networks, system-level support for information assurance, distributed collaborative systems and internet architectures. His recent project relationships with industries include IBM, AT&T, Philips and ITT. Besides industries, some of his research has been supported by grants and contracts from US federal government agencies (such as the Air Force Research Laboratory and Space Missile and Defense Command).

Kevin A. Kwiat has been a civilian employee with the US Air Force Research Laboratory in Rome, New York, USA, for over 25 years. He received his BS in Computer Science, his BA in Mathematics, his MS in Computer Engineering and his PhD in Computer Engineering from Syracuse University. He holds three patents. In addition to his duties with the Air Force, he is an Adjunct Professor of Computer Science at the State University of New York at Utica/Rome, an Adjunct Instructor of Computer Engineering at Syracuse University and a Research Associate Professor with the University of Buffalo. He completed assignments as an Adjunct Professor at the Utica College of Syracuse University, a Lecturer at Hamilton College, a Visiting Scientist at Cornell University and a Visiting Researcher while on sabbatical at the University of Edinburgh. His main research interest is dependable computer design.

Patrick Hurley is a Computer Engineer with the Air Force Research Laboratory's Information Directorate, Rome, NY, USA. He holds a Bachelor of Science in Computer Science from the State University of New York at Oswego and a Master's degree in Computer Science from the State University of New York Institute of Technology (SUNYIT). His research interests are in quality of service, cyber defense, survivability and recovery.

1 Introduction

In a real-time embedded system, it is required to move the data collected by devices from an external environment to the end-user. The data collection devices may be software and/or hardware modules interfacing with untrusted external parts of the system. Say, in one setting, wireless-connected devices sample the physical world data at regular intervals for surveillance purposes (as in sensor networks). In another setting, application agents running on system gateway nodes monitor the critical components of a large-scale system (as in infrastructure protection). Failures may arise however during data collection, because of the hostile nature of external conditions in an untrusted environment. The failures often manifest as data corruptions by malicious devices and timeliness violations in the processing and communication paths.

The data collection devices are replicated, to counter the effects of data corruptions and timeliness violations. The system employs some form of majority voting on the data fielded by various replicas (Jalote *et al.*, 1995) to decide on the delivery of a correct data to the end-user in a timely manner. The voting typically employs a two-phase mechanism: the proposal of candidate data by a device (first phase), and the collation of votes from the remaining devices by a centralised secure entity for decision-making (second phase). In this paper, we focus on the performance engineering of the two-phase voting protocol under various fault modes and network conditions and different application scenarios.

The data being voted upon can be large-sized and/or take a long time to be generated (such as images in a terrain surveillance system and transaction logs in an intrusion-detection system). Furthermore, the network may exhibit a wide range behaviours: a low message loss/delay in many situations and unexpected high loss/delay in other situations. The behaviour of faulty devices may itself be random, ranging between benign levels to malicious acts. Our goal is to engineer the voting protocols to achieve good performance despite the uncontrolled behaviours in the data collection infrastructure, while meeting the reliability requirements of data delivery needed of a high assurance setting.

The performance metrics we employ are the Data Transfer Efficiency (DTE) and the Time-To-Complete (TTC) a data delivery. DTE captures the network bandwidth wasted and/or the energy drain in wireless-connected devices. Whereas, TTC depicts the degradation in user-level Quality of Service (QoS) due to delayed and/or missed data deliveries. Both the metrics have a bearing on the user-level QoS: such as the operational life-time of wireless connected devices and the usefulness of delayed data to the application. Thus, improving DTE and TTC is a goal of our performance engineering exercise.

The voting protocol, in its basic form, requires the sending of consent and dissent votes (YES and NO) by devices about a data value being voted upon, to a central site *B*. See Figure 1. Suppose a data $X(v)$ proposed by voter v is put to vote. Thereupon, a voter v' sends YES or NO message to B based on whether its locally computed data $X(v')$ matches closely with $X(v)$ or not. Based on the YES and NO messages received from $\{v'\}$, B determines if $X(v)$ enjoys a majority consent for delivery to the user. The solicitation of votes from devices gets repeated until at least $(f_m + 1)$ consent votes are received – where f_m is the maximum number of devices that can be faulty. With N replica devices (where $N \geq 3$), we have $1 \leq f_m < \left\lceil \frac{N}{2} \right\rceil$.

Given the distributed nature of two-phase voting,[1] our protocol-level optimisations focus on reducing:

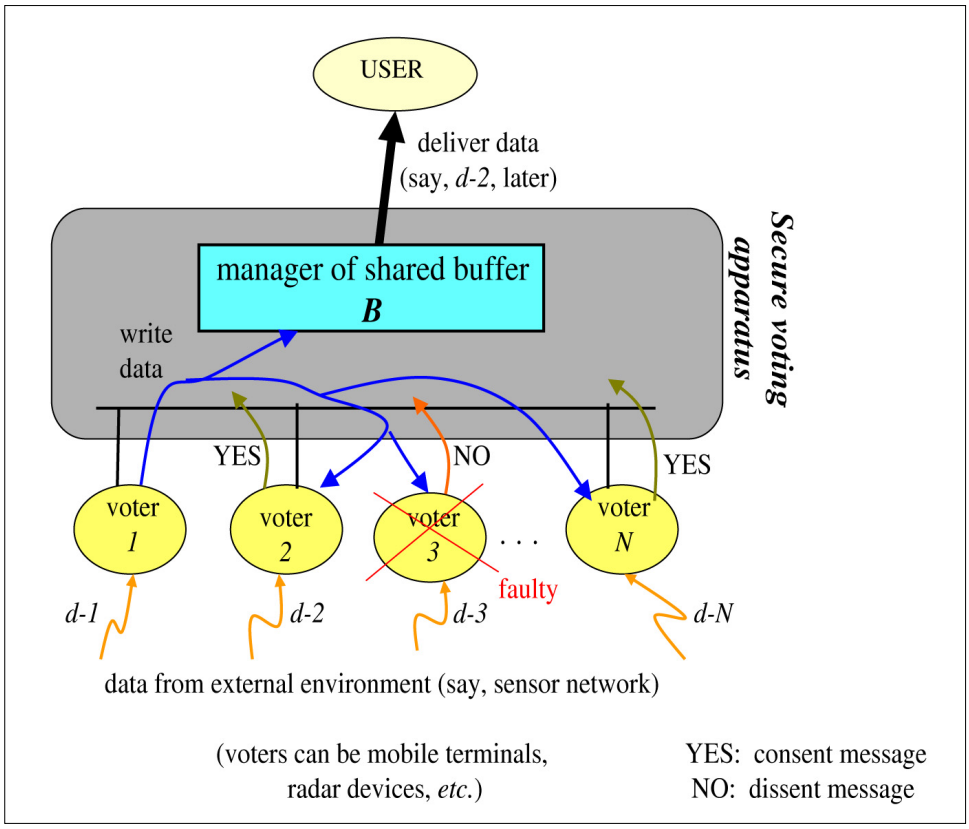
1 the movement of user-level data between voters

2 the number of voting actions/messages generated

3 the latency incurred by the voting itself.

To achieve (1), our voting protocol employs a 'listen-and-suppress' based coordination among voters to reduce the redundant data proposals that would otherwise be generated by various voters due to the spontaneity in their operations (such as another voter, besides $X(v)$, also proposing its data in the earlier scenario). For (2), the protocol employs a combination of selective vote solicitation and vote suppression, while ensuring that enough votes are available for decision-making. For instance, when $f_m \ll \frac{N}{2}$, the decision on delivering a correct data can be made with less message overhead. For (3), the protocol employs decentralised 'data comparisons' (*i.e.*, each voter compares its locally generated data with the candidate data), which allows the voters to decide on their votes in parallel.

Figure 1 Distributed voting protocol structure (see online version for colours)



While the techniques (1)–(3) are not orthogonal during a run-time execution of the voting protocol, they have varied levels of impacts on DTE and TTC depending on the operating conditions and the environment. The paper describes these optimisations, along with the experimental results from a prototype voting system. The voting protocol however satisfies the *safety* condition, namely, a corrupted data or a data that has exceeded its time

bounds never gets delivered to the end-user – even under strenuous failure scenarios. The paper also describes a case study of deploying voting in web server applications to access information repositories.

The paper is organised as follows. Section 2 gives a data-oriented view of the voting issues in distributed embedded systems. Section 3 identifies the role of replica voting in web services managing information objects. Section 4 describes the extensions needed of two-phase commit protocols to realise the voting functionality in asynchronous distributed settings. Section 5 describes the protocol optimisations to reduce the message overhead. Section 6 studies the voting protocols by software prototyping and performance analysis. Section 7 presents related works. Section 8 describes a case study on the data delivery performance of web App servers in the presence of replication and voting. Section 9 concludes the paper.

2 Data-oriented view of voting protocols

In this section, we describe why a content-dependent notion of device faults is necessary, and how this notion impacts the replica voting mechanisms.

2.1 Timeliness and accuracy of data

The data delivered to the user as representing an external event (or phenomenon) is associated with a timeliness parameter Δ. It depicts how soon the data produced by a sensor device should be delivered at the user since the occurrence of external event represented by this data (*i.e.*, life-time of data) (Kopetz and Verissmo, 1993).

The data generated by a sensor device may be somewhat inaccurate in content (relative to the actual reference datum) due to the inherent sampling errors in the sensing mechanism and/or resource limitations on the data pre-processing algorithms in the device (such as Central Processing Unit (CPU) cycles and memory sizes). Accordingly, the bit-level representations of data generated by two different devices may not show an exact match – even though the semantic contents of the data may be close enough to be validated as correct data (as is the case with non-numeric data such as images from remote cameras).

Consider, for example, the detection of an enemy plane flying at azimuthal location, say, 35.0^0. A radar unit may report detection at, say, 35.1^0 azimuth due to sampling error. This difference in the sampled value relative to an exactly sensed value (by an ideal device) results in a mis-match in their syntactic representations. The ‘data comparison’ procedure should however treat the two location reports as being the same in terms of their semantic contents (a numerical comparison operation on the sampled values is just a special case, as in Brooks and Iyengar (1998). The voting system should tackle the computational complexity involved in such a semantics-aware ‘data comparison’, and still deliver an accurate location report to the Command Center within a few seconds of the presence of enemy plane.

A device may control its selection of data processing algorithms and usage of computational resources, to provide results within a certain time deadline and content accuracy.

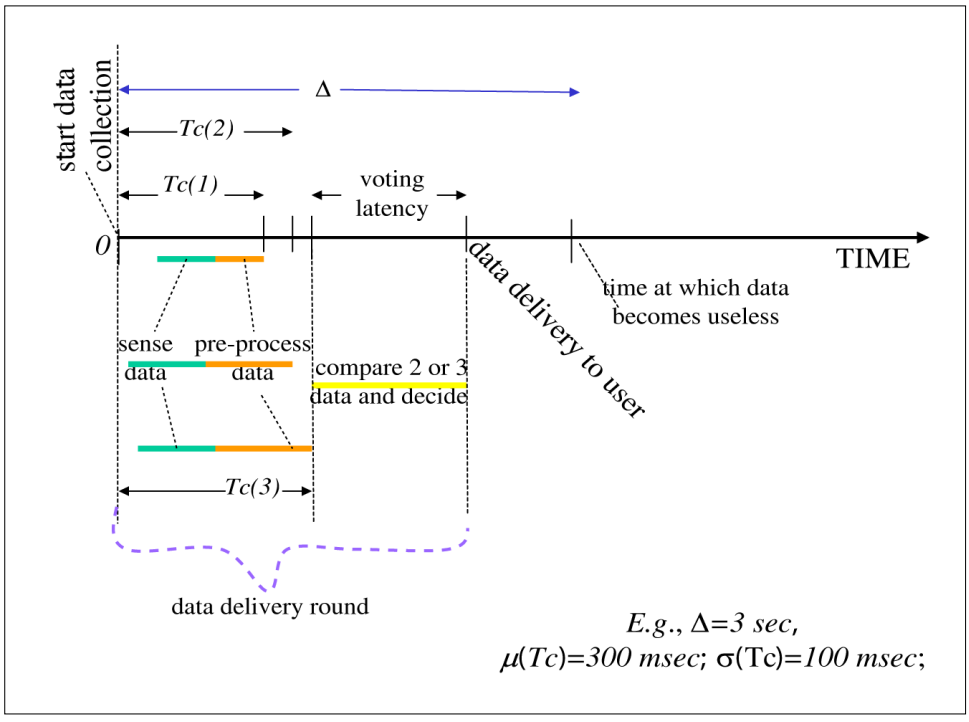
2.2 Semantics-aware data processing

Each voter pools its processing resources to compute the results from raw input data within a tolerable accuracy. Furthermore, the resource needs can be data-dependent. That the voters may employ different computational algorithms to process the input data brings in another issue as well, namely, the variability in computation times of devices.

Consider the data d_s from a device s that purports to represent the actual reference datum d^{ref} for an external object (or phenomenon) being sampled. That $T_c(d_s) < \Delta(d^{ref})$ depicts if the computation time $T_c(d_s)$ to generate d_s meets the timeliness attribute $\Delta(d^{ref})$ for the datum. That a data d_s should not deviate from its reference datum beyond a prescribed error limit $\epsilon(d^{ref})$ depicts a safety property associated with the delivery of d_s to the user, *i.e.*, $\| d_s - d^{ref} \| < \epsilon(d^{ref})$. In the previous radar example, ϵ may be set as, say, 0.2^0. The interpretation of ϵ when comparing the data from multiple devices requires a content-aware processing of the data.

The data from a non-faulty device always satisfies the timeliness and accuracy constraints. Whereas, a faulty device may violate the constraints in an unpredictable manner. In the earlier example of radars, a faulty radar may mis-report the location of an enemy plane as, say, 48.0^0, or, report a more accurate value of, say, 35.15^0 but after a couple of minutes which is too late to be of any use. In the presence of such faults, the voting protocol should validate a candidate data for its timeliness and accuracy before delivering it to the user. Figure 2 illustrates how the data processing delays in voting impacts the timeliness of data delivery.

Figure 2 Illustration of timeliness issues in voting (see online version for colours)



2.3 Protocol-level control of data delivery

From an algorithmic standpoint, the application environment may have at most f_m of the N voters as being faulty, where $N \geq 3$ and $0 < f_m < \left\lceil \frac{N}{2} \right\rceil$. This depicts the condition for determining if a candidate data is deliverable to the user.

A functional module B manages a buffer *tbu f* into which a device writes its data for voting. B resides within the secure enclave of voting machinery, and is securely connected to the user to whom a final result in *tbu f* gets delivered. The voter devices and B are connected through a secure multicast message channel, where communications are authenticated (with message signatures) and message contents are encrypted. Furthermore, the channels have certain minimum bandwidth guarantees, and enforce anonymity among voters. We assume that B is housed within a secure infrastructure that is immune from getting attacked.

A voter first proposes its data by a multicast-write into the remote buffer *tbu f*. From among multiple data items proposed, the buffer manager B selects a candidate data for voting, and solicits votes on this data. If a majority of YES votes occurs (or, $f_m + 1$ YES votes when $f_m < \left\lceil \frac{N}{2} \right\rceil - 1$), B passes on this data to the user. Otherwise, B selects a next candidate data for voting. If B cannot determine a majority before the deadline Δ, it discards the data (for safety reasons). Given the multicast transmission of data proposals across the voters and buffer manager B and the subsequent multicast of an index from B to refer to a data for which votes are solicited, it is guaranteed that the data getting the needed YES votes is already with B for delivery to the user. This functionality, combined with the use of secure channels, guarantees that any errors occurring during network transmissions will get detected.

In a real-time system where data may arrive continuously, the information loss caused by a missed data delivery can possibly be compensated by the subsequent data deliveries (Kopetz and Verissmo, 1993) (*e.g.*, periodic dispatch of terrain maps from a battlefield with adequate frequency). In this setting, the data delivery requirement can be relaxed: namely, the rate of missed data deliveries over an observation interval should not exceed a small threshold $\zeta(X)$, where $0.0 < \zeta \ll 1.0$.

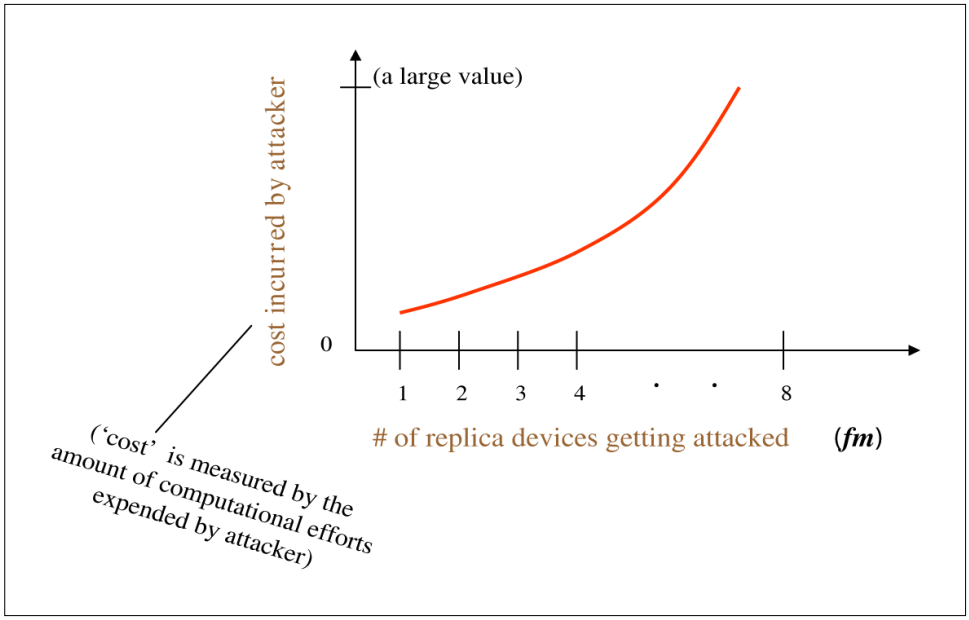
2.4 Partial synchrony and device heterogeneity

The 'partial synchrony' property of a system means that if an activity starts (say, a network message transmission), it will eventually complete in a finite amount of time. An upper bound on the completion time is however not known to the higher-layer algorithm running on the system (Castro and Liskov, 1999). In sensor networks for instance, the property manifests as follows.[2]

A non-malicious device will eventually report correct data and on time, if it has enough battery power. No device can be branded as faulty, in an algorithmic sense, unless it exhibits a sustained bad behaviour. A network channel that loses/delays messages intermittently will eventually transmit a message successfully. And, enough number of sensor devices remain in the field so that a management entity assigns the task of replicated data collection to N devices.

The devices are made heterogeneous to minimise the chance of all of them getting attacked at once: such as running on different CPUs, running under different operating systems and programming languages, and/or implementing different computational algorithms to process data (Forrest *et al.*, 1997). Here, the cost incurred by an intruder to attack the various devices increases drastically with respect to the number of devices targeted for attacks. See Figure 3. The drastic increase in intruder's cost arises from the need to coordinate the attacked devices and synergise their damaging effects on the data delivery system. Furthermore, the increased risk of exposure of the intruder as more devices are targeted for attacks also contributes to the higher cost of attacks faced by the intruder. Given the finite amount of resources at the disposal of an intruder and the incentives for initiating attacks, we thus believe that only a small number of devices will actually be attacked. Given a f_m, the degree of replication N can then be chosen such that $f_m \ll \frac{N}{2}$.

Figure 3 Empirical view of cost incurred by intruder to attack devices (see online version for colours)



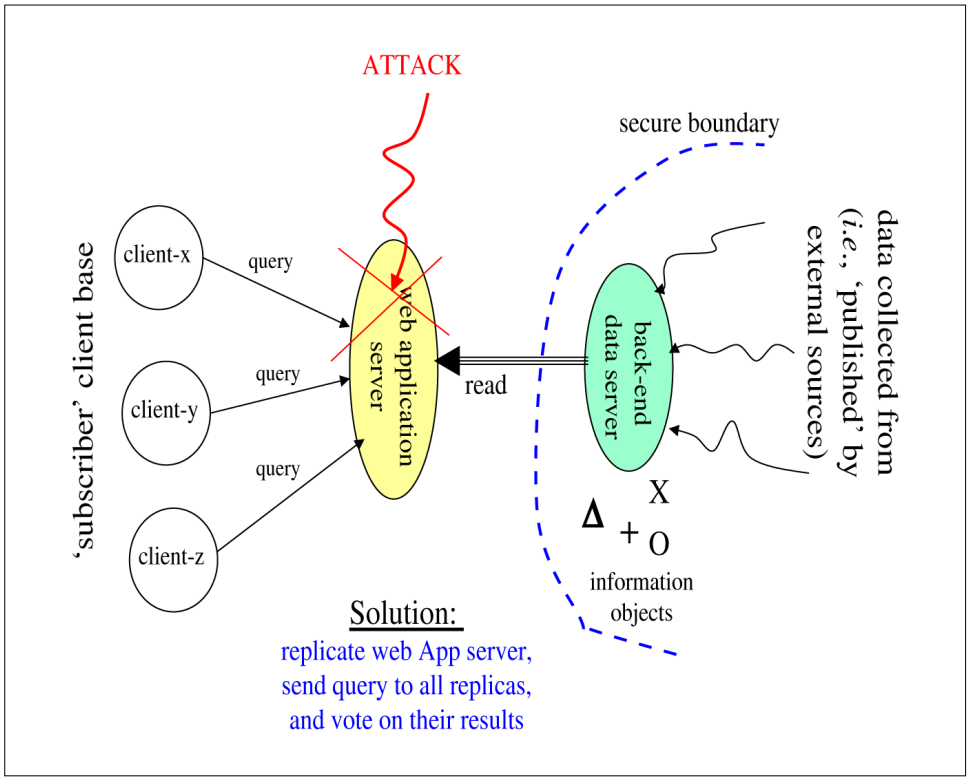
The induced heterogeneity among devices for attack-resistance manifests however as asynchrony in their computations, *i.e.*, randomness in their times of data generation. This aspect is captured, at the voting protocol level, by the partial synchrony property – and the corresponding liveness assertions in terms of ζ.

We now identify a deployment scenario of voting in representative applications, namely, a web service to access information objects. It allows us to clearly delineate the external interface to the voting machinery from the protocol-internals.

3 Deployment scenario of voting in web services

A web service is provided by one or more App servers that process client queries about the information objects maintained by back-end data servers (we use the terms 'App servers' and 'web servers' interchangeably). In this section, we describe the application characteristics that impact the deployment of voting-based solutions. See Figure 4.

Figure 4 Current model of web service to access information repositories (see online version for colours)



3.1 Pre-processed information objects

The information objects are often structured pieces of pre-processed data about the application's external environment (*e.g.*, data collected from target tracking sensors in a military application, market indices prepared by trend watchers in a stock market application).[3] A web service allows client applications to access the repository of information objects in the form of queries. For example, an object may be the radar image of a battle terrain, and a client query may by to find out if a plane has been detected in the image. Multiple client queries may be posted concurrently on a web service, wherein the processing actions on these queries share the computational and network resources at the server end.

A comprehensive operational system is the Joint Battlespace Infosphere (JBI) at the US Air Force Research Laboratory Information Directorate (AFRL) to support an Air Operations Center (Ramseyer *et al.*, 2006). The JBI provides an extensive query interface for 'subscriber' clients that allows multi-clause conjunctive queries on objects. In a terrain surveillance application for example, a query such as: 'is there a plane in a given range and azimuth of the terrain?', is a conjunctive clause specifying the interval in which the range and azimuth values should fall. The object update interface for 'publisher' clients expects that the objects written into have already been checked for correctness through other means (say, to prevent malicious code from being carried as an object data).

In the commercial world, IBM's Infosphere Master Data Management Server (IBM Software Group, 2008) supports business process managements by allowing the creation of and access to multiple information objects (say, about customers, products, and accounts) through web interfaces. An example application is the online reservation of passenger travel by an airline, where the information objects in back-end store are the flight schedules, price and availability data, reservation records, partner airlines data, and the like. The web service is a front-end to the airline database, often obtaining aggregated travel information therefrom in response to client queries (*e.g.*, a listing of multi-airline travel itineraries within a certain price range).

As can be seen, web service based access to information objects[4] has become a paradigm to handle the complex cyber data spaces in military, industrial, and business applications.

3.2 Mobile access to information objects

Often, the clients are mobile, being implemented on PDAs and Pocket-PCs held by human users and connected to the data and information servers through wireless networks. Access to the information servers is via a web interface that is part of a publish-subscribe paradigm of information management. Some clients may be beyond the Line-of-Sight (LOS) range from the information objects maintained by the back-end data servers, often, due to radio range limits of the mobile clients. Here, the web service may be viewed as extending the LOS range of clients by moving the pre-processed information physically closer to the clients over a data transport network (the RF information are part of the meta-data in the repository).

As an example, consider a military application setting for battlefield information management through a web service where the clients are the soldiers deployed in a field, aircrafts monitoring a terrain, and naval ships engaged in maritime surveillance. Typically, all combat units (*i.e.*, soldiers, ships, and aircrafts) over a given geographic region subscribe and receive Situational Awareness (SA) data, say, about the location of friendly forces in that region. The SA data may be, say, annotated and geo-referenced still images from video feeds and metadata-tagged full motion video snippets stored in a central repository of terrain surveillance data.[5] Disadvantaged clients may request only meta-data objects, and download either still images or video snippets based on specific filter criteria (carried in the client queries). Asynchronously generated alerts can also be configured within the client subscriptions to determine if other friendly forces come into visible range.

Though a web service improves the LOS capabilities of mobile clients, the additional nodes and links in the extended transport network may themselves induce failures – in addition to the server vulnerabilities caused by the inherently soft nature of interfaces to the web service. In this paper, we focus on web server vulnerability issues due to attacks and the underlying voting-based solutions (the QoS parameter ζ depicts the query drop rate due to failures).[6]

3.3 Attacks on web App servers

Web application servers are often easy targets for attacks because they are publicly and freely accessible (but for, say, simple password-based authentications) and act as gateways to the back-end data server(s) where the information objects are kept. An example is the SQL injection attack (Giannoulis, 2008) where the attacker inputs a SQL-based search field in its query, and if the query is accepted, the attacker in effect gains access to the information parts stored in the back-end data server. Another form of attacks is that a malicious code in the web application server can corrupt the results of a client query on the information objects maintained by back-end data servers (we assume that the data servers are secured from external attacks through other means).

To minimise the effects of attacks on web servers, the servlet processing a client query is replicated on multiple machines. These servlet replicas independently process the queries and compute the results. A voting apparatus placed in the client-interface to the web service compares the results generated by the replicas and votes on them. A result that is in the majority is then treated as the correct result for delivery to the client (if the timing constraint Δ is also met).

Our study considers the voting protocol based solutions to deal with the malicious faults that may occur when processing the client queries (*i.e.*, the reads from an information repository) by servlet replicas. Here, we assume that the back-end data servers maintaining the information repository are well-secured and that the information updated by 'publisher' clients are sanitised through other means (*i.e.*, the fault-tolerance mechanisms for write operations on the repository by 'publisher' clients are outside the scope of our paper).[7] Accordingly, the type of web services we consider in the paper separates, in an architectural way, the query operations (*i.e.*, the reads) from the update operations (*i.e.*, the writes) on the back-end data servers – such as the JBI at AFRL.

The issues of system asynchrony (at network/process levels), corruption of query results, and state-machine failures of system processes that arise during a query processing call for our generalised solution based on replica voting to achieve fault-tolerance in a scalable manner. In this light, other existing works also suggest the use of replication-based fault-tolerance, *al beit*, for grid environments (such as the eDemand project at University of Leeds) (Townsend and Xu, 2004) or with different solutions (such as optimistic quorums to handle updates and queries) (Malek *et al.*, 2005).

In the light of afore-described data characteristics and operating environments of distributed information systems, we outline the voting protocol extensions next.

4 Functional elements of voting protocols

In this section, we identify the main elements of the two-phase voting protocol from a performance standpoint (*i.e.*, to reduce the message overhead and data delivery latency). We assume that message loss in the network is not high in normal cases.

4.1 Determination of majority from votes

Two-phase voting protocols require the sending of consent and dissent votes (*i.e.*, YES and NO messages) about a data value being voted upon, to a central vote collator. The protocol determines a majority based on the number of YES (or NO) votes from among the responses received. That only the votes received are considered in the decision (instead of requiring all the N votes) guarantees liveness in the presence of send-omissions of faulty devices and network message loss – see Prisco *et al.* (2000). We refer to the protocol as M2PC (modified two-phase commit based voting).

Note, a non-faulty device X that does not have its data locally computed yet votes NO for a data v from another voter X' currently put to vote. This is because X does not have the local context yet to determine if v is good data or bad data – and hence X votes NO for safety reason.

The partial synchrony property of the system allows B to determine if the data v being voted upon can be safely delivered. Here, the safe delivery of v requires B knowing that v is indeed good data, *i.e.*, B seeing at least f_m devices, excluding the proposer of v, cast YES votes for v. In a case where v cannot be safely delivered, B solicits a next data proposal from the remaining voters as a new *iteration*, while declaring the proposer of v as ineligible to propose again (the latter action prevents livelocks in the presence of persistent bad proposals by a malicious device). The liveness guarantee is that a good data will be identified in at most $2f_m + 1$ iterations of the data delivery *round*. If the time to decide on a correct data delivery (*i.e.*, TTC) exceeds Δ, B does not deliver any data to the end-user. The non-delivery of data in a voting round constitutes a data miss at the user level.

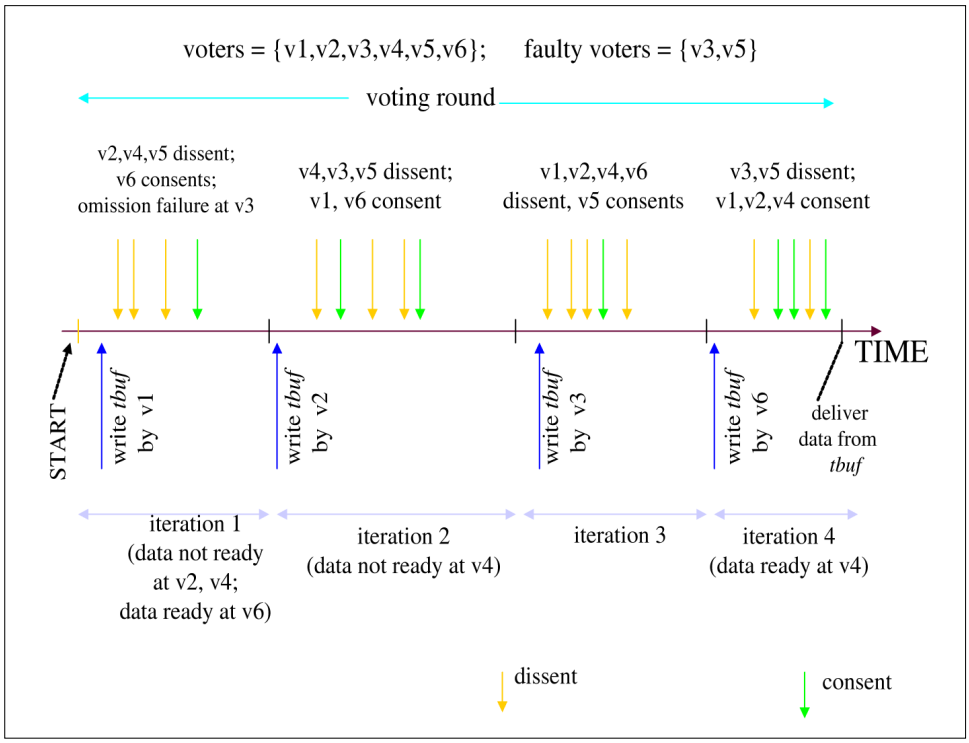
The M2PC protocol incurs a worst-case message complexity of $\mathcal{O}(N^b)$, where b is a constant: $1.0 \leq b \leq 2$. The actual value of b depends on the environment, such as the extent of voter asynchrony $\sigma(T_c)$ and the number of faulty voters f.

Figure 5 shows a scenario of asynchronously executed vote solicitation steps (*i.e.*, iterations) in a data delivery round. For the computation asynchrony shown with $N = 6$ and $f = 2$, B cannot safely deliver a data in the first three iterations even though good data are voted upon twice – they do not muster the 4 YES votes needed for a majority. Because there is at least one non-faulty voter, namely, v_4, that does not have its locally computed data ready yet. And, a safe data delivery occurs only in the fourth iteration when v_4 also has its data ready – assuming that TTC < Δ. The scenario shows 7 YES votes and 12 NO votes over four successive iterations in a voting round.

In contrast, a centralised placement of 'data comparison' functions, as would be needed in coordinator based voting schemes (Jalote *et al.*, 1995), requires shipping the large-sized non-numeric data to the central node B (*e.g.*, terrain images from remote cameras). Here, the data movement overhead over the network is: $(f_m + 1)$ in the best case and $(2f_m + 1)$ in the worst case. By careful engineering of the M2PC scheme on the

other hand, a voting incurs just one multicast data transfer in the best case,[8] with the semantics-aware 'data comparisons' carried out locally at the voters in parallel – thereby reducing the drain on battery energy and the voting latency.

Figure 5 A scenario of asynchronous voting with M2PC protocol (see online version for colours)



For the purpose of this paper, we assume that Δ is quite large compared to the network and computation delays. This means that M2PC has almost-zero data misses, because the timing constraints on data delivery will rarely be violated (even under a high degree of asynchrony in the system).

5 Additional protocol mechanisms

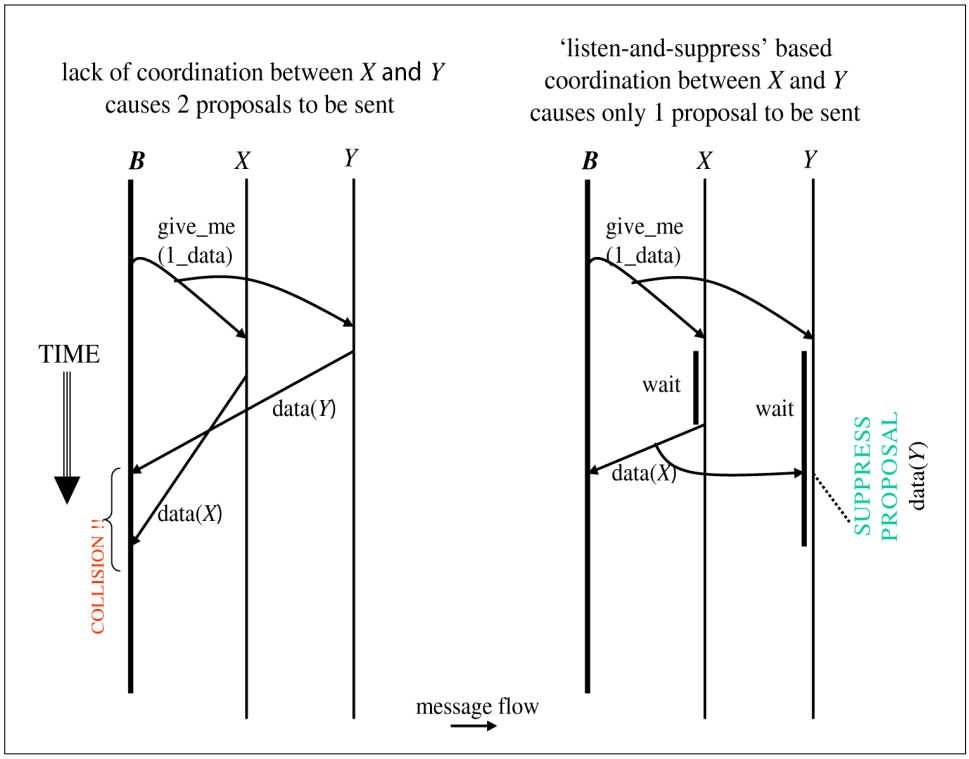
In this section, we describe the various functional elements that are layered on top of the basic voting protocol, to enhance the performance and robustness.

5.1 Optimisations during data/vote solicitations

One mechanism is the suppression of redundant data proposals from the various competing voters that independently attempt to multicast their proposals. In the absence of knowledge about which of the eligible devices will compute their data sooner and which devices are faulty, *B* cannot pre-select the devices to solicit data proposals without

affecting performance. So, we employ a distributed approach where a device *X* waits for a random interval of time before multicasting its proposal message (carrying the computed data). If *X* receives a proposal from another device *Y* while waiting to propose, *X* suppresses its attempt to propose because a proposal from *X* will be redundant in the face of the proposal already sent by *Y*. But, *X* votes on the data of *Y* (or any other data) when the latter is put to vote by *B*. The above 'random-wait-before-propose' and 'listen-and-suppress' techniques reduce the network overhead when large-sized data proposals are involved (such as images). See Figure 6 for an illustration.[9]

Figure 6 'Listen-and-suppress' of redundant data proposals (see online version for colours)



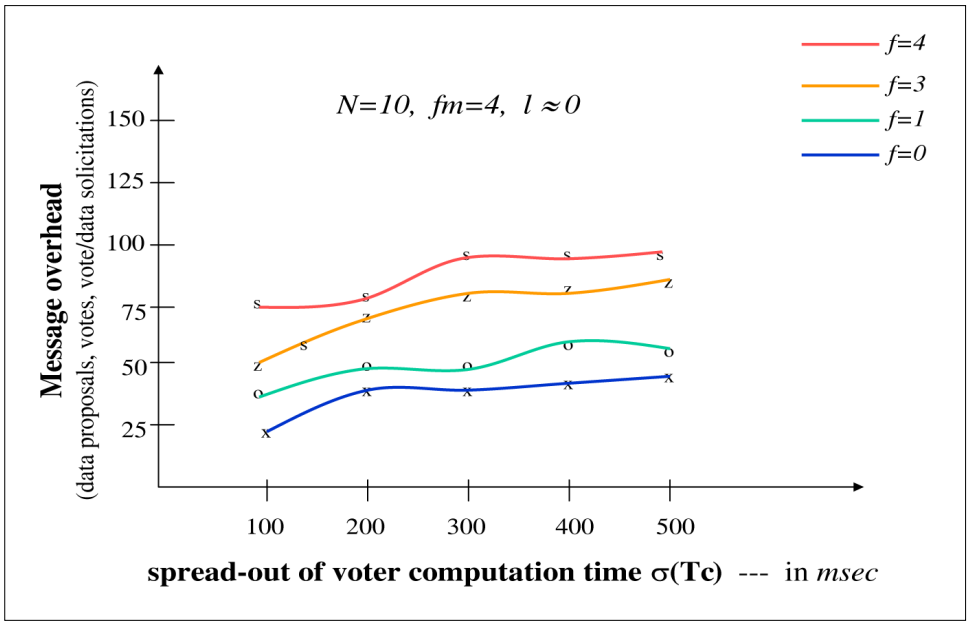
Another mechanism is the wait of *B* for a suitable amount of time before soliciting the votes for a data proposal *Y*. The wait time is based on the system asynchrony parameters, namely, $\mu(T_c)$, $\sigma(T_c)$, and $\overline{T}_d$. The deferment of vote solicitation allows a majority of the voters to have their locally computed data ready, so that they can vote YES/NO for *Y* in a single iteration. This reduces the amount of vote message exchanges that may otherwise be needed, *i.e.*, when *B* can decide on a data delivery only after multiple iterations. Referring to Figure 5, the best point for *B* to start the vote solicitation is the time at which the four non-faulty voters (v_1, v_2, v_4, v_6) have their locally computed data ready. Given the voter asynchrony compounded by random network delays, *B* determines the best time point for vote solicitation based on the statistical parameters of T_c.

Protocol-level details of the above optimisations are beyond the scope of our paper (see Ravindran *et al.*, 2008).

5.2 Study of the effects of computation asynchrony

We have conducted experiments to study the impact of computation asynchrony on the amount of message exchanges incurred to effect a data delivery to the end-user (without the 'deferred vote solicitation' mechanism). The graph in Figure 7 shows the experimental results on message overhead in the M2PC mode[10] for $\sigma(T_c)$ ranging from 100 *msec* to 500 *msec* in a case of [$N = 10$, $f_m = 4$, $l \approx 0$].

Figure 7 Impact of computation asynchrony on message overhead (see online version for colours)



Setting aside the details of experiments (described later in Section 6), the results show an increase in the message overhead as $\sigma(T_c)$ increases, for any given f. For $f = 0$ (which depicts the absence of faulty behaviour), the message overhead is about 25 for $\sigma(T_c) = 100$ *msec*, and is about 40 for $\sigma(T_c) = 500$ *msec*. The 2 and 3.5 voting iterations needed respectively in these cases arise due to the increasing number of voters who have not yet computed their local data when B solicits the votes for a candidate data. For the cases of [$f = 1$, $f = 3$, $f = 4$], the faulty data proposals and the asynchrony-induced aborts of good data proposals have a compounded effect of increasing the number of voting iterations – and hence the message overhead: [38, 50, 75] for $\sigma(T_c) = 100$ *msec* and to [48, 76, 87] for $\sigma(T_c) = 500$ *msec*.

5.3 Selective vote solicitation

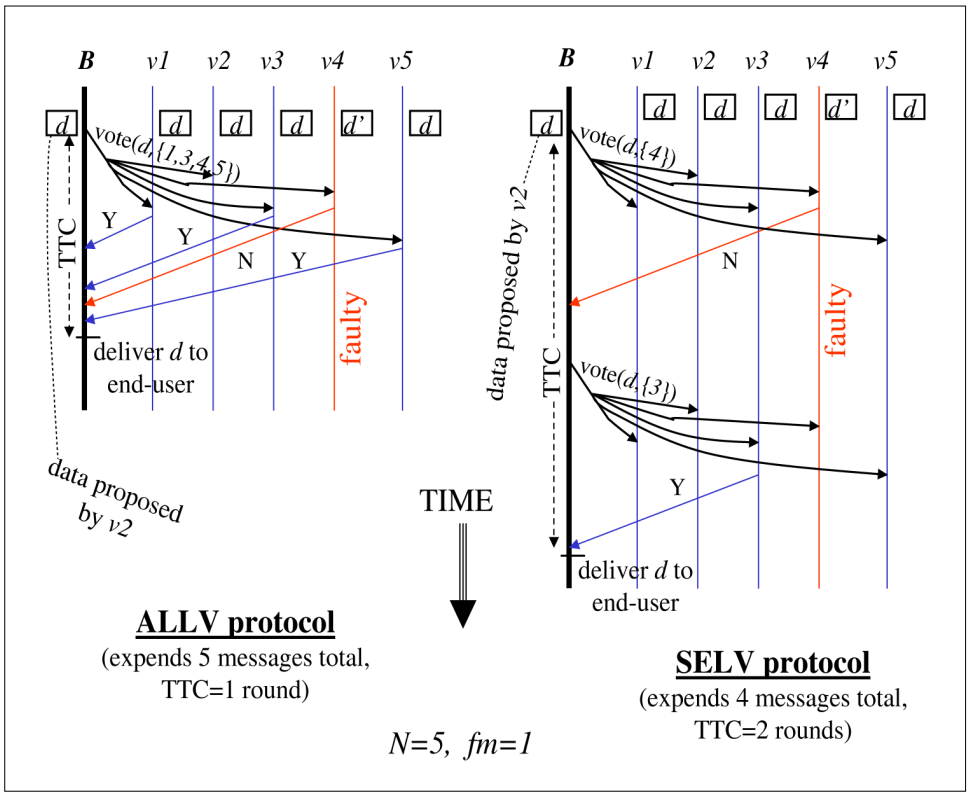
We also implement a *selective vote solicitation* scheme, referred to as SELV, in which B multicasts a message asking for votes only from a selected subset of voters in the group. This is in contrast from the basic vote solicitation mechanism in two-phase voting, denoted as ALLV, that generates vote messages from all the $(N - 1)$ voters for a

candidate data *d* being voted upon. The performance-engineered SELV mechanism reflects an optimistic premise about the sporadic occurrence of faults in the normal case, namely: the number of faulty voters is very small (*i.e.*, $f_m \ll \left\lfloor \frac{N}{2} \right\rfloor$) and a faulty voter exhibits faulty behaviour only sporadically.

With a candidate data being good most of the times, it is likely that the required f_m YES votes will be received in the first round of selective vote solicitation itself. In a case where there are not enough YES votes, *B* excludes the first round of voters from its solicitation list, and then seeks votes from one or more of the remaining voters based on the number of remaining YES votes needed. This proceeds until *B* actually receives the required number of YES votes.

Figure 8 illustrates the ALLV and SELV schemes for a case of $N = 5$ and $f_m = 1$. As can be seen, the ALLV scheme always incurs a message overhead of $(N - 1)$ and completes in one round of voting. Whereas, the SELV scheme incurs a lower message overhead but may go through more than one round of voting. For the SELV scheme to be effective, the buffer manager *B* should wait until all the $(N - f_m - 1)$ non-faulty voters are likely to have their data ready. This increases the vote solicitation time, and hence increases the latency in data delivery to the end-user.

Figure 8 A sample illustration of the ALLV and SELV schemes (see online version for colours)



In our implementation of SELV, B specifies the subset with a bit-map (of size N bits), where each bit corresponds to a distinct voter and the ON/OFF of a bit indicates if the corresponding voter should send its vote to B or not. B may use any additional knowledge available to it in setting the bit-map to reduce the control message overhead – such as which devices have a smaller $\mu(T_c)$. Given that the network can lose messages, B may repeatedly seek votes only from the voters whose responses are not seen in the previous steps – until $f_m + 1$ YES votes or $N - f_m$ NO votes are seen.[11]

We now describe a software prototyping and performance study of our optimised voting protocols.

6 Prototype implementation and analysis

We have simulated the M2PC protocol on a local network of SUN-UNIX workstations interconnected by IP multicast message transport and UDP-based unicast transport over Ethernet. The buffer manager and the voters are implemented as UNIX threads, communicating with one another through IP and UDP sockets. We also have an implementation of the protocols on SHARP-SL3780 Pocket-PCs running LINUX.

6.1 Simulation of voting protocols

To simulate faults, a voter is marked as GOOD or BAD, with a BAD voter randomly injecting faults: such as omission failures, data corruptions, aggressively proposing data to outrun the GOOD voters, and other state-machine violations. Even under intense fault scenarios, the protocols function correctly, namely, always deliver only good data. Since our goal is primarily to demonstrate the performance aspects, the study has focused more on the normal case protocol operations. The input parameters are f_m, T_c Δ, and l.

We incorporate the variance arising from domain-specific computations in the performance model of M2PC by simulating their effects with randomly set computation times and data sizes (in the ranges 20–30 *msec* and 0.2–50 *kbytes* respectively). Data are sent using IP messages, with the UNIX system internally segmenting the messages into smaller sized packets, if necessary, during transport. The message loss naturally occurring in the UDP/IP network is in addition to the simulated message loss inflicted on the voting protocol.

We also simulate the effect of voter asynchrony during ‘data generation’ in our protocol studies. The variability in ‘data comparison’ times is incorporated by using a random number that sets the wait time of processes around a large mean based on the data sizes and the word-by-word matching times required on a SUN-Ultra-10 CPU.

The time-to-complete a voting activity is a performance metric visible to the user applications. The rate of missed voting rounds (*i.e.*, how often the data delivery to the user fails) is another metric of interest to capture the effect of message loss and delays on enforcing the timeliness constraint Δ for data delivery. The system overhead is measured by the amount of messages exchanged during a protocol run (on average).

6.2 Performability measures

We measure the control message overhead (CNTRL), data transmission overhead (DAT), and TTC for various cases. The data transfer efficiency is then given by:

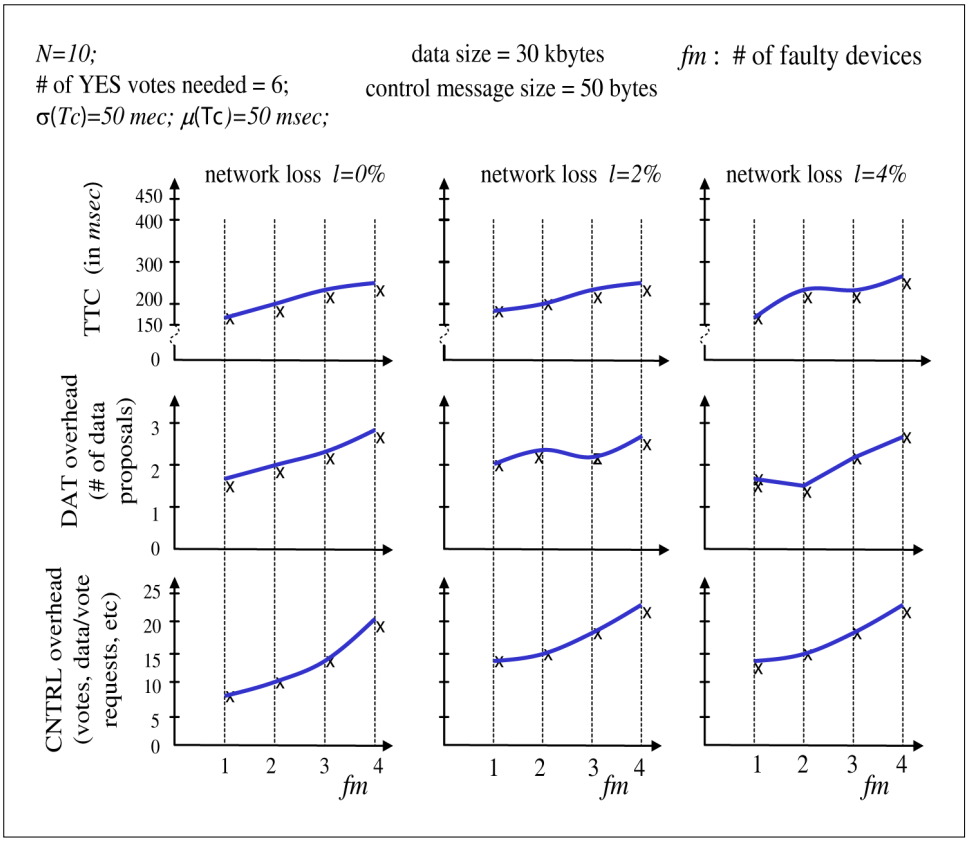
$$\text{DTE} = \frac{\text{size(data\_delivered)}}{\text{DAT} \times \text{size(data)} + \text{CNTRL} \times \text{size(control\_message)}}.$$

Typically, the data size ranges between 100 *bytes* to 40 *kbytes*, based on the type of data voted upon. Whereas, the control messages (such as the YES/NO votes from devices and the vote requests from *B)* are 50 *bytes* long.

For a data size of 30 *kbytes* and [$\sigma(T_c)$ = 100 *msec*, N = 10, f_m = 4, $l \approx 0$], TTC lies in the range 175–210 *msec* (we set f = 4 to simulate the extreme case). The DAT overhead is about 2.5 and CNTRL = 18.5.

For analysis purposes, we treat (f_m, l) as external parameters that cannot be controlled but strongly impact the data delivery performance of M2PC. To study their effects, comprehensive experimental results are shown as graphs in Figure 9, for the cases of f_m = 1, 2, 3, 4 (with N = 10 and $f = f_m$) and l = 0%, 2%, 4%. The other parameters are set as: $\sigma(T_c)$ = 50 *msec*, and $\mu(T_c)$ = 50 *msec*. We corroborate the experimental results based on the functional elements of M2PC, and their impact on TTC, DAT and CNTRL.

Figure 9 Performance results on M2PC (see online version for colours)



6.3 Observations on performance results

The TTC depends on a variety of factors: the network message delay T_d and the 'data comparison' time T_{cmp}. In our experiments on SUN-UNIX over Ethernet, $\mu(T_d) = 6$ *msec* and $\sigma(T_d) = 2$ *msec*. We set $T_{cmp} = 100$ *msec* as the wait time of a voter, to simulate a 'data comparison' operation. So, *B* can decide in about 120 *msec* (when message loss rate $l \approx 0$). This, combined with the time taken for 'data proposal' phase and the 'smart wait' of *B* for $\frac{\mu(T_c)}{2}$ in vote solicitation phase corroborates the results: TTC ≈ 180 *msec* in the case of $l \approx 0\%$. The dependence of TTC on *l* is captured in the 8% increase of TTC when *l* is increased to 2%.

The DAT overhead shows an increase when f_m becomes large. This is due to the increasing number of faulty voters who propose their data. The increase in DAT from about 2 for $f_m = 1$ to about 3.5 for $f_m = 4$ corroborates this. Note, this DAT overhead is distinct from the CNTRL overhead that occurs in a subsequent vote solicitation phase.

The CNTRL overhead shows an increasing trend with f_m. This is because of the increased number of aborts (due to a bad data proposal or an 'early-bird' proposal), with a consequent revoting in the later iterations. As for the impact of *l*, M2PC expends quite a number of control messages. This is due to the persistence of M2PC in determining if a candidate data is good or bad by message retransmissions.

For loss rates exceeding 10%, the data delivery latency becomes quite high due to the loss of data and/or many YES/NO votes in the network and the consequent delay in validating a data. This can result in a data miss if the application prescribes a tight deadline Δ on data delivery.

6.4 Analytical study of selective vote solicitation (SELV)

We have carried out a probabilistic analysis of the SELV mechanism with two simplifying assumptions:

1 the network does not lose packets

2 all the non-faulty devices have their locally computed data ready to enable them cast votes.

We assume that the f_m faulty voters are randomly placed among the *N* voters. Let $Q_{(M, f, k, s)}$ denote the probability of receiving *k* YES votes from a voter solicitation list of size *s* selected from an ensemble of *M* voters that contains *f* faulty voters, where $1 \le f < M$, $0 \le k \le s$, $1 \le s \le (M - f)$, and $1 < M \le N$. Let $\bar{L}_{(x, y, z)}$ denote the average number of message exchanges incurred in order for *B* to receive *z* YES votes from an ensemble of *x* voters that contains *y* faulty voters, where $1 \le z \le (x - y)$, $0 \le y \le f_m$, and $1 \le x \le (N - 1)$. Based on permutation analysis, we have:

$$Q_{(M, f, k, s)} = \frac{C_k^{M-f} . C_{s-k}^{f}}{C_s^M},$$

where $0 \le k \le f$ and $0 \le (s - k) \le f$. Using this basic probability formulation, we have the following recurrence relation for message overhead:

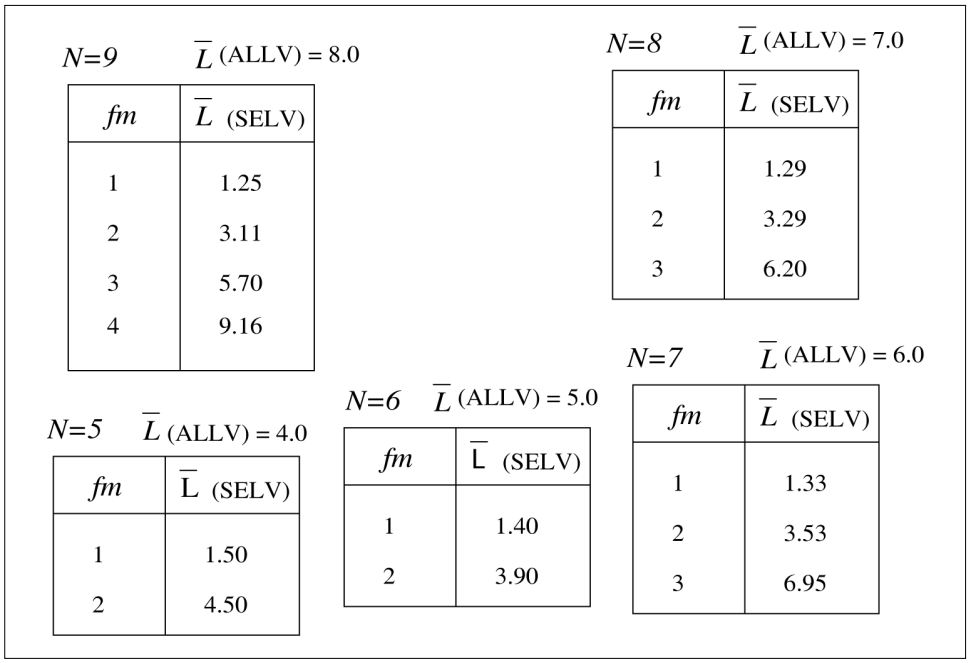
$$\bar{L}_{(x,y,z)} = Q_{(x,y,z,z)}.z + \sum_{t=z-y}^{z-1} Q_{(x,y,t,z)}.(z+1+\bar{L}_{(x-z,y-z+t,z-t)}) \text{ for } z \geq y > 1;$$

$$= Q_{(x,y,z,z)}.z + \sum_{t=0}^{z-1} Q_{(x,y,t,z)}.(z+1+\bar{L}_{(x-z,y-z+t,z-t)}) \text{ for } 1 \leq z < y;$$

with the terminal conditions being: $\bar{L}_{(x,1,1)} = Q_{(x,1,1,1)}.1 + Q_{(x,1,0,1)}.3$ (*i.e.*, for the case of $y = z = 1$) and $\bar{L}_{(*,0,z)} = z$. Along similar lines as above, a recurrence relation for the TTC of the voting, can also be formulated in terms of the number of distinct message exchanges to solicit votes.

The table in Figure 10 gives the message overhead and the TTC results for various values of N and f_m. In the case of $N = 9$, the SELV protocol incurs a lower message overhead for $f_m \leq 3$; but for $f_m = 4$, the ALLV protocol incurs a lower overhead. Likewise, in the case of $N = 7$, the SELV protocol incurs a lower message overhead for $f_m \leq 2$; but for $f_m = 3$, the ALLV protocol incurs a lower overhead.

Figure 10 Results on voting protocol performance and end-user QoS



The message overhead results need to be combined with the TTC results in evaluating the overall protocol cost, by employing a user-supplied QoS utility function. The latter typically assigns a lower utility for higher TTC values. Since the TTC for SELV protocol is always higher than the TTC for ALLV protocol, we expect that the overall cost break-even point will occur at lower values of f_m than the results obtained with message overhead alone.

As can be seen, the various protocol elements of M2PC interact with one another in improving the overall performance of data delivery to the end-user.

7 Related works on voting

Consensus protocols dealing with 'Byzantine Generals Problem' (Lamport *et al.*, 1982) (where failures may be arbitrary) can, in principle, be used for distributed voting, say, to tolerate a failed voter sending conflicting information to the other voters. These protocols are however quite expensive for deployment in power-constrained sensor network settings, and often do not scale well – due to the need to realise (f_m + 1) disjoint message paths on a shared network infrastructure and the assumptions about synchronous behaviour of system elements.

The modified form of two-phase commit we consider in the paper, referred to as M2PC, has the vote collation and the decision about commit relegated to *B*, while the problem-specific (and potentially compute-intensive) data interpretation for the purpose of casting votes is relegated to the replicas themselves.[12] Given that a majority of the voters are non-faulty, the M2PC can arrive at a safe commit decision on the data written into the buffer in a timely manner.

A recent proposal described in (Castro and Liskov, 1999) attempts to realise a practical approach to byzantine fault-tolerant systems by relaxing the need for synchronous behaviour of system elements (to carry out Byzantine voting) and by using non-forgeable signatures in messages. This approach still requires that $f_m \leq \left\lfloor \frac{N-1}{3} \right\rfloor$. Our approach, in contrast, employs a centralised secure entity, namely, *B*, and resorts to the use of authenticated communication between protocol entities, thereby avoiding the expense of a full Byzantine fault-tolerant system.

Other voting protocols studied elsewhere advocate a centralised or semi-centralised placement of the data comparison function. The centralised placement, as in coordinator based voting schemes (Jalote *et al.*, 1995), incurs a large amount of data movement from voters to the central node. Likewise, the 'witness'-based voting scheme (Du *et al.*, 2003) employs a semi-centralised structure with an intermediate fusion node *S* to collect the data of 2*f* other voters. *S* then appends its own result to the data collected for onward transmittal to a base station. The latter then performs data comparisons.[13] Both the schemes incur data movement overheads in the range $\mathcal{O}(f)$ to $\mathcal{O}(N)$.

In contrast, our topological structure is decentralised, with the voters comparing the data themselves. With multicast transmission of data, our approach reaps a higher performance: due to the parallelism in data comparisons at various voters and the avoidance of unnecessary data movements. The worst case data movement overhead is $\mathcal{O}(f)$.

Thus, our approach offers attractive benefits for sensor networks, given that data sets may be large and contain non-numeric data (*e.g.*, terrain images generated by radar scans).[14] The benefits are further accentuated in wireless network settings when the power savings accrued by reductions in data transmission overhead and the highly dynamic nature of environment are taken into account.

8 A case study of voting in web services

In this section, we undertake a case study of voting based incorporation of fault-tolerance in the processing of web service queries initiated by client applications on the information objects maintained by back-end data servers.

8.1 Processing of client queries

A client query, say, expressed as a predicate of conjunctive clauses expressed in a XML-like language, is first processed by the server for syntactic checks. A query often prescribes logical conditions that need to be evaluated by comparing the actual data contained in the information repository with the reference data and/or constants indicated in the various clauses. The data involved in a predicate evaluation can be of complex type (*e.g.*, terrain images). After syntactic checks on a client query, the server dispatches a *servlet* to actually process the query. The servlet carries out two operations:

1 process the raw data in the repository in response to the client query about an event of interest

2 notify the result of processing to the client as to whether the prescribed event has occurred.

The processing of client queries is subject to the issues posed by the system-level asynchrony (such as the queuing of queries at servlets and the interleaving of multiple concurrent queries).

If the servlet gets attacked, the processing of client queries can be flawed, which leads to incorrect results returned to the client. It is this kind of failure that server replication and voting purports to deal with.

8.2 Attack models on web servers

We assume that the repository is not corrupted and does not contain malicious data.[15] Given that the native information objects are not corrupted, the faults that can occur are only in the query servlet processing engines (such as a malicious code that alters the processing of raw data and/or distorts the outcome of processing). Typically, the information repository and the applications that access the repository are in different administrative domains – and hence are subject to different security and fault-tolerance procedures.

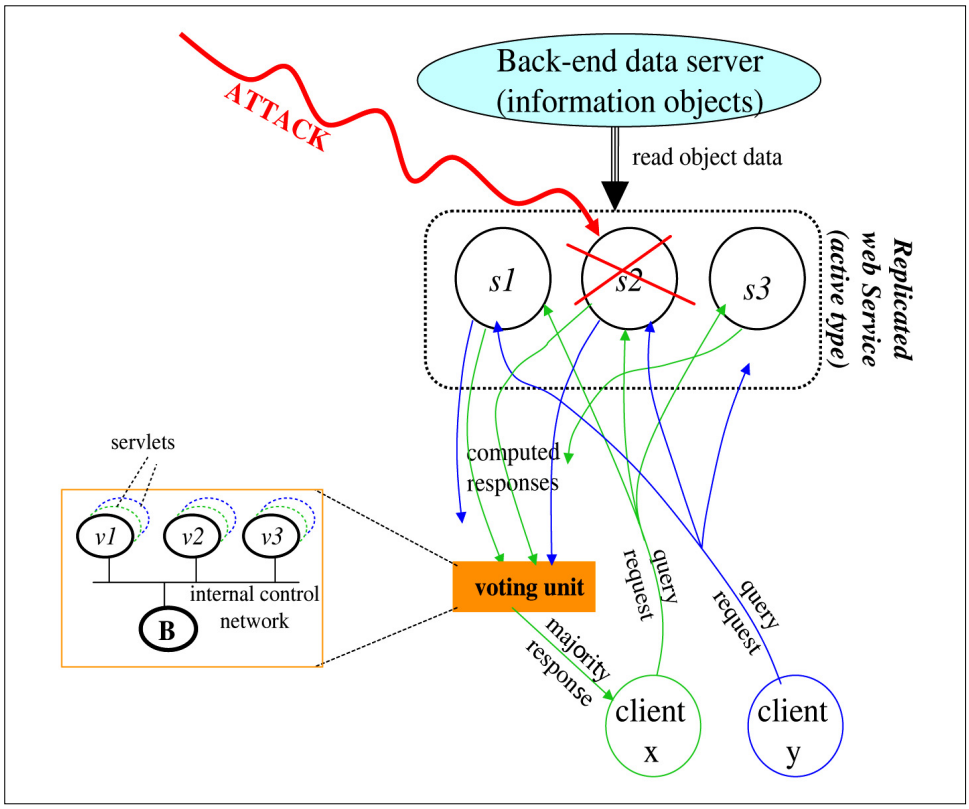
Consider, for example, the repository of stock market indices (such as DJIA, S&P500 and SENSEX) that are updated in response to the changes in the industry profiles and market conditions. Different stock broker applications may access this market index information to predict the changes in stock prices in their own ways. Managing the repository of stock market indices falls in a distinct administrative domain to provide, say, fault-tolerance and performance in the update activities. The broker applications are vulnerable to data corruptions that may occur in thequery paths, and hence incur the cost of replicated query processing to enhance the reliability of information access. Different broker applications may need different levels of data resilience, and are willing to incur the cost of achieving the desired level of resilience.

The attack model described in Section 2.4 (cf. Figure 3) holds for servlet failures in web service settings.

8.3 Voting-based structure of client-interface to web services

A separate servlet instance is created that runs on behalf of the web server to process the queries initiated by a client. A servlet may be replicated to run on different machines for fault-tolerance (and for performance too, in some cases). A client query is multicast to all the replicas for processing. The multiple query results are then voted upon to decide on a correct result for delivery to the client. Figure 11 illustrates the system structure.

Figure 11 Servlet replication and voting for query processing (see online version for colours)



The replication, combined with the voting, purports to provide fault-tolerance in the query processing. If p_a is the probability of a servlet getting attacked, the probability of a query failure at the client level (*i.e.*, the probability that a query is not guaranteed to yield correct results) is:

$$P_{qf}(replica) = \left[1 - \sum_{f=0}^{\lceil \frac{N}{2} \rceil - 1} C_f^N \, p_a^f . (1 - p_a)^{N-f}\right],$$

where N is the number servlet replicas handling the given type of client queries. Note, the case of no replication has a query failure probability P_{qf} (*single*) = p_a. For *e.g.*, P_{qf} (*replica*) = 0.00856 for $N = 5$ and $p_a = 0.1$, *i.e.*, a five-fold replication increases the data reliability by a factor of 12. The degree of replication N is determined by the cost of replication weighed against the need for higher data reliability.

Note, P_{qf} depicts the failure rate of the voting system itself due to an intruder attacking more than a majority of the N devices. In contrast, the QoS parameter ζ depicts the query drop rate when the voting system successfully operates (*i.e.*, less than a majority of the N devices fail and a data miss is detected at run-time due to the expiry of delivery deadline Δ).

8.4 Performance impacts of servlet replication

Multiple servlets may be running in the system, possibly on different machines, for performance reasons. For instance, different servlets may process queries that have different response time requirements by allocating appropriate CPU, memory and bandwidth resources from a shared resource pool. Such a software structure allows the servlets to enforce different levels of query response behaviour (*i.e.*, QoS) for the various clients.

Besides fault-tolerance, servlet replication also offers performance improvement by parallel processing of the client queries. This however requires a load-balancing mechanism to route the concurrent client queries to various servlet replicas.

Our case study focuses on the performance impacts of providing fault-tolerance by servlet replication and voting. The underlying mechanism we consider is the processing of a query by multiple replicas of the servlet that handles the given query type and then the voting on their responses to decide on a correct result for delivery to the client. The voting protocol instance working on a data is modelled as a task dependency graph, where the various sub-tasks in the graph are distinct protocol activities carried out in a certain order (say, the data solicitations/proposals and the generation of YES/NO votes). The sub-tasks of various concurrent voting activities get queued at system resources (network, CPU, and memory storage), and are executed as per the task dependencies.

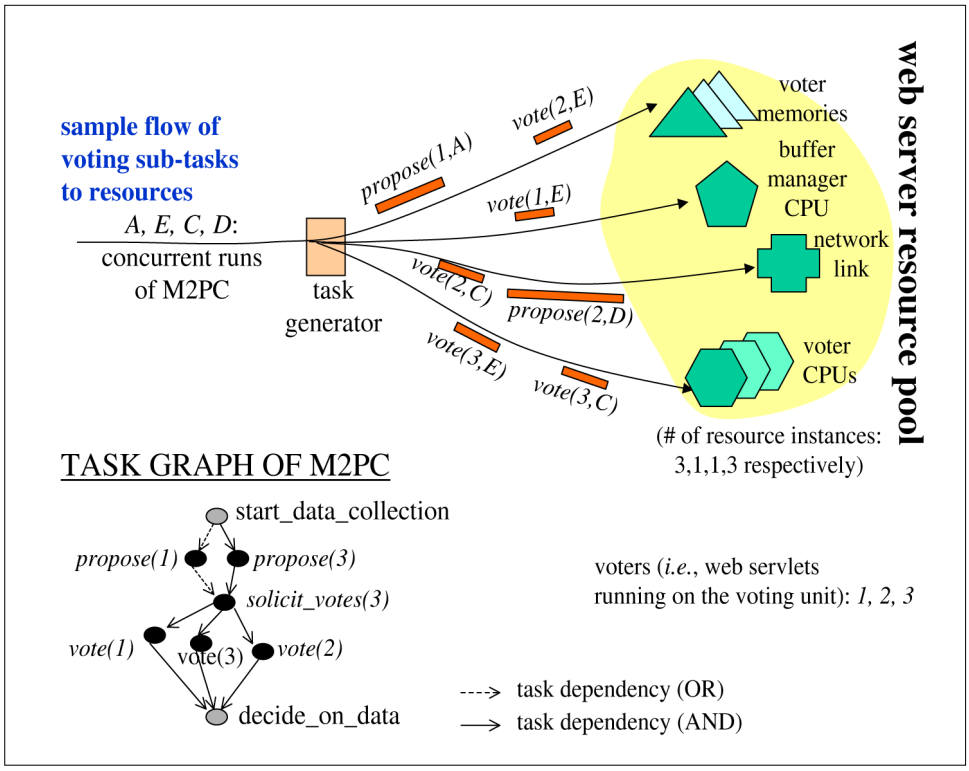
Figure 12 shows the inter-play between the parallel processing of client queries for performance and the synchronisation of query results by replica voting for fault-tolerance.

The queuing delays at system resources increase the data access latency experienced by clients. So, a resource scheduler manages the flow of sub-tasks through various resources in order to achieve a desired QoS and performance.

8.5 Web service performance metrics

We study the data delivery performance of a web service in the presence of servlet replication and the voting-based support mechanisms. The study adopts the QoS and performance models developed elsewhere for App Servers implementing business processes and transactional queries (Garcia *et al.*, 2008).

Figure 12 Concurrent execution of query processing and voting tasks (see online version for colours)



The performance metrics that are meaningful in web services are the Query Response Time (QRT) and the Control Overhead (COV) to deliver query results. QRT consists of the time spent in processing a query and in voting on the query results. Whereas, COV consists of the CPU cycles expended by the servlet (CPU_s), storage accesses on the back-end data server (DSK_d), and message exchanges with other servlets to carry out the voting (MSG_v). In terms of the protocol parameters in our voting model, we have:

$$QRT = \overline{T_c} + T_{vlat};$$
$$COV = k_1.CPU_s + k_2.DSK_d + k_3.MSG_v;$$

where k_1, k_2, k_3 are normalisation constants. As seen, the voting function in the client-server interface determines in part, but not entirely, the performance at the web service level.

It is reported in Ramseyer *et al.* (2006) that the (mean) query response time in the AFRL JBI setting is about 150 msec when concurrently processing 50 single-clause queries. With the incorporation of servlet replication and voting and the underlying multicast communications needed therein, we expect the QRT to increase by at least the TTC estimated in our experiments.

A comprehensive performance evaluation of a voting-based replicated web server system is a part of our further research.

9 Conclusions

We considered replica voting protocols for reliable data dissemination in an environment where malicious failures can occur at the data collection devices (compounded by message loss/delays in the transport network).

The functional extensions needed to the basic form of two-phase majority voting are the provisioning of additional protection layers to sanitise the voting decisions in the presence of sustained message loss. For instance, if the time elapsed becomes too high since the start of a data collection, the data delivery is aborted due to real-time deadlines. The required augmentations to the M2PC protocol to increase its robustness and performance were also studied.

The paper described protocol-level optimisations needed to implement M2PC in a data collection system: a 'listen-and-suppress' based reduction of redundant data proposals and a 'selective vote solicitation' to reduce the flurry of vote message exchanges. The important goal is to ensure the integrity of data delivery to the end-user in the presence of data corruptions and other faults in the data collection system. A performance thrust is to reduce the message overhead, which reduces the network bandwidth consumed for user-level data delivery (and the power drain on wireless sensor devices where appropriate). Suitable performance metrics were also identified.

Our voting system is adaptive to deal with the various failures, benign or malicious, occurring in the data collection environment (as in military surveillance systems and infrastructure protection systems). As a case study, the paper described how the voting system may be deployed in Web App servers to access information repositories (*e.g.*, JBI at AFRL).

Acknowledgements

Acknowledgements are due to Dr. Ali Sabbir of the Independent University, Bangladesh and Mr. Jiang Wu of the City University of New York (CUNY) for their roles in the earlier stages of the project, particularly, the experimental studies on voting protocols. Acknowledgements are also due to Mr. Mohammad Rabby of CUNY for implementing the protocols on SHARP Pocket-PC based wireless network platforms.

References

Babaoglu, O. (1993) 'Non-blocking commit protocols', in S. Mullender (Ed.) *Distributed Systems*, Chap. 7, Addison-Wesley Publ. Co.

Brooks, R.R. and Iyengar, S. (1998) 'Sensor fusion and approximate agreement', *Multisensor Data Fusion*, Prentice-Hall Publ.

Castro, M. and Liskov, B. (1999) 'Practical Byzantine fault tolerance', *Proc. 3rd Symp. on Operating Systems Design and Implementation*, New Orleans, Los Angeles, February.

Du, W., Deng, J., Han, Y.S. and Varshney, P.K. (2003) 'A witness-based approach for data fusion assurance in wireless sensor networks', *Proc. IEEE-GLOBECOM'03*, December.

Forrest, S., Somayaji, A. and Ackley, D.H. (1997) 'Building diverse computer systems', *Proc. 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, IEEE.

Garcia, D.F., Garcia, M., Garcia, J. and Entrialgo, J. (2008) 'A simulation model to develop QoS control strategies for application servers', *SCS Journal on Simulation*, February–March, Vol. 84, Nos. 2–3, pp.75–88.

Giannoulis, P. (2008) 'Finding and blocking web application server attack vectors', *SearchSecurity.com*, http://'searchsecurity.techtarget.com/tip/0,289483,sid14gci1252706,00.html.

IBM Software Group (2008) 'IBM Infosphere master data management server: technical overview', White paper, February.

Jalote, P., *et al.* (1995) 'Atomic actions on decentralized data', *Fault-tolerant Systems*, Chap. 6, John-Wiley Publ. Co.

Kopetz, H. and Verissmo, P. (1993) 'Real time dependability concepts', in S. Mullender (Ed.) *Distributed Systems*, Chap. 16, Addison-Wesl. Co.

Lamport, L., *et al.* (1982) 'The Byzantine generals problem', *ACM Trans. on Prog. Languages and Systems*, Vol. 4, No. 3, July.

Malek, M.A., Ganger, G., Goodson, G., Reiter, M. and Wylie, J. (2005) 'Fault-scalable Byzantine fault-tolerant services', *Proc. 20th ACM Symp. on Op. Sys. Principles, ACM SIGOPS*, Brighton, UK, October.

Pollack, S. and McQuay, W.K. (2005) 'Joint battlespace infosphere applications using collaborative enterprise environment technology', in R. Suresh (Ed.) *Proc. SPIE-5820, Defense Transformation and Network Centric Systems*, Belligham, Washington.

Prisco, R.D., Lampson, B. and Lynch, N. (2000) 'Fundamental study: revisiting the Paxos algorithm', *Theoretical Computer Science*, Vol. 243, pp.35–91.

Ramseyer, G.O., Yan, L.K. and Linderman, R.W. (2006) '100X Joint Battlespace Infosphere (JBI)', In-house Interim Technical Report, AFRL-IF-RS-TR-2006-265, Air Force Research Laboratory Information Directorate, August.

Ravindran, K., Wu, J., Rabby, M., Sabbir, A. and Kwiat, K.A. (2008) 'Performance engineering of replica voting protocols in high assurance data collection systems', *Proc. Intl. Conf. on COMmunication Systems SoftWAre and MiddlewaRE (COMSWARE)*, Bangalore, India, January.

Townsend, P. and Xu, J. (2004) 'Replication-based fault-tolerance in a grid environment', Technical report, School of Computing, University of Leeds.

Notes

1 That not all the $(N - f_m)$ non-faulty devices need to agree on the data being delivered makes the voting protocol a weaker instance of the two-phase non-blocking consensus protocol (Babaoglu, 1993).

2 In the internet for example, an IP packet retransmitted many times will eventually reach the intended receiver, but the sender may not know about the number of retransmissions needed.

3 The information repository is updated with new objects or object modifications by 'publisher' clients that process raw data from the external environment for writing into the repository (*e.g.*, sensors deployed in a field) (Pollack and McQuay, 2005). 'subscriber' clients query the repository to identify objects of interest via read operations on the data servers (typically, the 'subscriber' clients far outnumber the 'publisher' clients). The system employs a service-oriented architecture, exporting enquiry and update operations on the master data.

4 A web service embodies computational processing on the information objects in back-end store (namely, object transformation and fusion) when responding to client queries – unlike a simple non-computational web browser interface to the objects. A web service access may however involve a browser that runs on the customer computers, PDAs, kiosks, and the like (*e.g.*, travel agents and passengers accessing an airline reservation web service).

5 See AFRL Technical Report entitled as: *Use Case for Information Management*, AFRL-RI-RS-TR-2008-214 (authors: T. Clark and A. Kwiat), for a description of mobile web-based applications in military settings.

6 The web service also carries out a variety of presentation-related functions such as video scaling and layering, transcoding, and formatting to display the processed objects for dissemination by the mobile clients – say, to adapt to the network conditions such as limited bandwidth availability and packet loss. Such transport layer issues are also relevant in a system-level implementation of the voting-based mechanisms for dependable content delivery to a client.

7 The notion of correct read of data from the information repository by a 'subscriber' client is relative to what data has been written into the repository by a 'publisher' client. The data read can at best be as good as the data that was written into earlier. The scope of our paper is that even if an object in the repository is already corrupted, a fault-tolerant read of this object will still be deemed as correct if the returned value is consistent relative to the (corrupted) data originally written into.

8 M2PC incurs (f_m + 1) data movements only in the worst case. The substantial savings in data movements over the centralized scheme outweighs the $\mathcal{O}(N^b)$ short YES/NO messages needed in the M2PC scheme.

9 Recall that the use of multicasts for data proposals over secure channels allows detecting network errors that may occur during message transmissions – cf. Section 2.3.

10 A message may be the data proposal from a voter, the YES/NO vote cast by a voter, the vote/data solicitation from *B*, and the like.

11 The buffer manager *B* can exploit its knowledge about the status of faulty devices as gleaned in the earlier iterations/rounds, to selectively solicit votes from a small set of non-faulty devices. This allows a data delivery to occur in a relatively short time (say, with even less than f_m YES votes), in the case of emergency access to data. Since the knowledge about which devices are faulty can be imperfect, there is however a risk of delivering incorrect data if one or more of the voters selected in the bit-map are faulty. This risk of bypassing the standard voting procedure should be evaluated using external mechanisms in the light of emergencies that require quick access to data.

12 This is unlike the current transactional models of voting where the 'data comparison' operations are often trivial: such as each replica deciding as to whether a transaction (*i.e.*, a sequence of operations) on shared data should be committed or aborted based on local conditions.

13 *S* may itself be a source of malicious failure, thereby requiring complex protocol mechanisms to deal with this possibility (*e.g.*, the base station rotating the fusion role of *S* across different voter nodes).

14 In both the centralised and semi-centralised structures, the computational burden of semantic interpretation of data is placed at the central node – which may not be feasible in certain applications.

15 The mechanisms to secure the information written into the repository against malicious 'publisher' clients are outside the scope of our paper.