Replica Voting Based Data Collection: Adaptive Protocols and Application Studies

Mohammad Rabby, Kaliappa Ravindran
The City University of New York
New York, NY 10031, USA
Email: mfrabby@yahoo.com; ravi@cs.ccny.cuny.edu

Kevin A. Kwiat
Air Force Research Laboratory
Rome, NY, 13441, USA
Email: kwiatk@rl.af.mil

***Abstract*—**
Data collection in a hostile environment requires dealing with malicious failures in the sensing devices and underlying transport network: such as data corruptions and message timeliness violations. Functional replication is employed to deal with failures, with voting among the replica devices to deliver a correct data to the end-user. The paper describes the protocol-level design issues for voting with low message overhead and delivery latency under various failure scenarios. The well-known 2-phase voting protocol is employed as a building-block, with different protocol variants dynamically selected based on the network conditions and device-level fault severity. A case study of replicated web services is also presented to illustrate the usefulness of our dynamic adaptation mechanism.

Keywords - Majority voting, sensor replication, malicious faults, voter asynchrony, adaptation to environment.

I. Introduction

In a distributed information system, it is often required to move the data collected from an external environment to the end-user (e.g., market indices for stock-purchase, terrain images for surveillance). Problems arise however due to failures occurring during data collection, because of the exposure of data collection devices and/or the data flow paths to hostile external conditions. The failures often manifest as data corruptions by malicious devices and timeliness violations in the processing and communication paths.

The devices are often replicated to mask out failures, with some form of majority voting employed to reach agreement on the data fielded by various replicas [6]. In this paper, we focus on the voting protocol mechanisms to deal with data corruptions and enforce a timely delivery of correct data to the end-user. We employ a variant of the 2-phase majority voting to validate the data fielded by replicated devices.

The 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

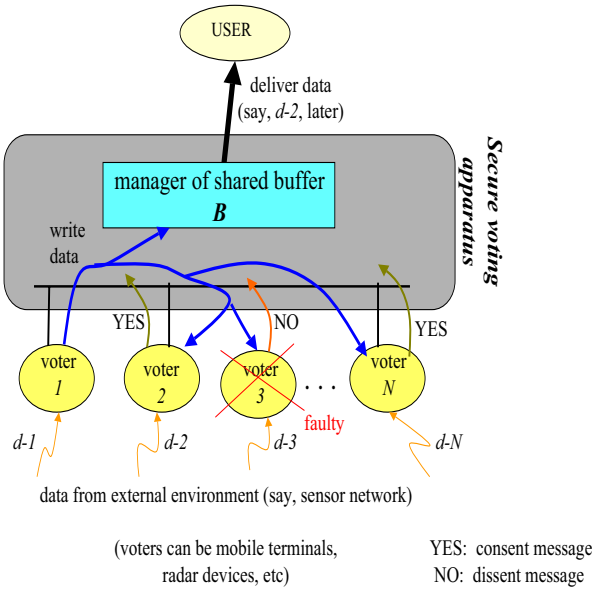

Fig. 1. Distributed voting protocol structure

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 $1 \leq f_m < \lceil \frac{N}{2} \rceil$. When $f_m \ll \frac{N}{2}$, the decision on delivering a correct data can be made with less message overhead.

Replica voting has been studied by many researchers in the area of database, file, and web servers: primarily to achieve service-level fault-tolerance and performance. In the context of sensor-based embedded systems however, two important issues come to the forefront. First, the limited battery power of sensor devices imposes a severe constraint on the message-level overhead of voting protocols and the data exchanges between voter devices. Second, the widely varying channel characteristics of communication paths between the devices and the end-user (such as packet loss) can make protocol performance unpredictable. Given these issues, it is necessary that voting protocols are adaptive to the changing operating conditions, while providing a dependable data delivery to the end-user with minimum power and bandwidth consumptions.

The paper describes our extensions to the voting protocol mechanisms to deal with the various failures occuring in the

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-109-0

environment (such as device attacks and network packet loss). Our extended protocol strives to reduce the number of control messages and application data exchanged between various nodes during voting. This becomes important when large-sized sensor data such as images are considered.

The paper is organized as follows. Section II gives a data-oriented view of replica voting. Section III describes 2-phase commit protocols for voting in resource-scarce systems, with adaptation to the environment conditions. Section IV describes a case study of voting protocols in replicated web services. Section V concludes the paper.

II. Data-oriented view of voting protocols

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

A. Timeliness and accuracy of data

The data delivered to the user as representing an external event (or phenonmenon) may be associated with a timeliness parameter Δ. It pertains to how soon the data produced by a sensor device be delivered at the user since the occurrence of external event represented by this data (i.e., life-time of data). The Δ parameter impacts the data delivery mechanisms embodied in the voting protocol.

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 CPU cycles and memory sizes). Accordingly, the bit-level representations of data generated by two different devices may not exactly match — even though the semantic contents of these data may be close enough that they can be declared as representing the same object (as is the case with non-numeric data such as camera images).

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. Despite this difference in the sampled value and the consequent mismatch in its syntactic representation, the 'data comparison' procedure should treat the two location reports (provided by different devices) as being the same in terms of their semantic contents. The voting system should tackle the computational complexity involved in such a 'data comparison', and still deliver an accurate location report to the Command Center within a few seconds of the presence of enemy plane.

The data generated by 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 unit may report the location of enemy plane as, say, 48.0^0, or report a more accurate value of, say, 35.15^0 but after a couple of minutes. 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

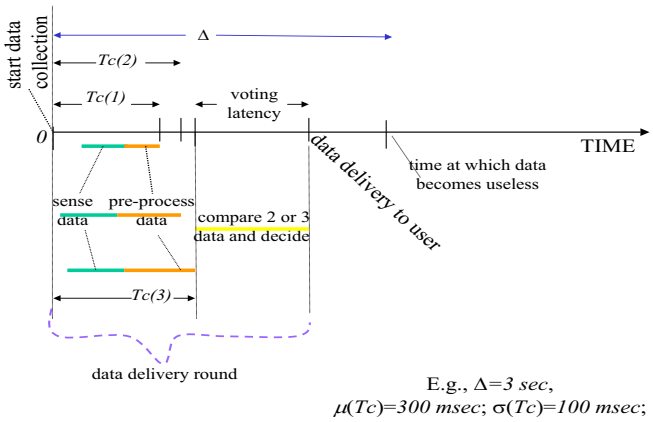

Fig. 2. Illustration of timeliness issues in voting

processing delays incurred for voting impact timeliness at user-level. The device computation time T_c to generate data and the network delay for vote collation are other influential factors.

B. 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 $0 < f_m < \lceil \frac{N}{2} \rceil$. This depicts the condition for determining if a candidate data is deliverable to the user.

A functional module B manages a buffer $tbuf$ 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 $tbuf$ gets delivered. The voter devices and B are connected through a secure multicast message channel, where communications are authenticated, 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 $tbuf$. From among multiple data items proposed, the buffer manager B selects a candidate data for voting, and then solicits votes on this data. If a majority of consents (i.e., $f_m + 1$ YES votes) are received from the voters, 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 data delivery deadline Δ, it discards the data (for safety reasons).

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

C. Partial synchrony in communications

Partial synchrony in 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

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-109-0

algorithm running on the system [9]. In our data collection system, the 'partial synchrony' property manifests as follows.

A non-malicious device will eventually report a correct data (the device is assumed to have adequate computational resources: such as CPU cycles, bandwidth, and battery power). No device can be branded as faulty, in an algorithmic sense, unless it exhibits a sustained bad behavior. A network channel that loses messages intermittantly will eventually transmit a message successfully. And, adequate number of replica units remain in the field so that a management entity assigns the task of data collection to N units for voting purposes.

In the light of above data characteristics and operating environments, we outline the voting protocol extensions next.

III. Augmentations to voting protocols

In this section, we augment the 2-phase voting protocol with an implicit voting mode to enhance the performance under normal cases, i.e., when packet loss in the network is small.

A. Explicit determination of majority

2-phase voting protocols require the sending of explicit consent and dissent votes (i.e., YES and NO messages) about a data 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 [5]. We refer to the protocol as M2PC (modified 2-phase commit).

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

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

In contrast, a centralized placement of 'data comparison' functions, as would be needed in coordinator based voting schemes [6], 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 1 multicast data transfer in the best case[1], 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.

B. 'Voter silence' as consenting vote

The large number of YES/NO message exchanges in M2PC raises scalability concerns with respect to the size of voter complex N. An implicit-consent based voting protocol [3] solves the scalability problem. In this protocol, B solicits only dissents, if any, from the voters for a candidate data. The large number of YES responses is avoided by having a consenting voter X not send any message and B treating the 'lack of message from X' as an implicit notification of consent. The protocol, referred to as IC-M2PC, rests on the premise that a large majority of voters are non-faulty (i.e., $f \ll \frac{N}{2}|_{N \gg 3}$) and message loss is sporadic in the normal case.

When f is close to $\lceil \frac{N}{2} \rceil - 1$ and/or many dissent messages are lost or excessively delayed, it is possible for a faulty data to be delivered inadvertantly. This is because B construes the non-receipt of a (dissent) message from a voter X as a consent by X. The motto is: NO NEWS IS GOOD NEWS — which is an optimistic one ! The likelihood of an undetected mis-delivery of data (due to missed and/or lost dissents) should be kept to a small minimum — say, $< 10^{-2}$. Where a 100% guarantee in the integrity of data delivery is required, a supplementary mechanism should be resorted to by the voting layer to ascertain the voters' intent through other means of low overhead probing. The choice is based on how often the above cases arise and the relative design complexities.

The message overhead of IC-M2PC may be given as $\mathcal{O}(N^{c'})$, where $0.0 < c' \leq 1.0$ in the normal case and $c' < c$ in the worst case (i.e., $f \rightarrow \lceil \frac{N}{2} \rceil - 1$). Here, $c' \rightarrow 0^+$ captures the case of voting on exactly one data item that requires just one message exchange (from the voter who proposed the data). And, $c' \rightarrow c^{(-)}$ captures the case of voting on multiple data items, with all but the last item getting discarded.

Figure 3 illustrates IC-M2PC based voting relative to M2PC, with timing scenarios. As can be seen, the YES vote generated by M2PC does not occur in IC-M2PC. But, IC-M2PC takes longer to complete the voting because of the wait for a full timeout period of T_w to determine if there are any dissents; whereas, M2PC may complete the voting quicker if $f_m + 1$ YES votes are received well before the expiry of timeout T_w.

C. Vote history based sanity check

From an algorithmic standpoint, the network-induced message loss or excessive message delays are indistinguishable from send-omissions of voters. This uncertainty has different implications on M2PC and IC-M2PC: an inability of M2PC to establish a majority consent/dissent if a large number of the voter responses were lost, whereas IC-M2PC may inadvertantly decide to deliver a bad data by believing that the voters whose dissent responses were lost have consented. To deal with this problem, IC-M2PC employs a history vector based mechanism to sanitize the commit decisions.

Here, B solicits the history of how each voter had cast its votes in the last K rounds of voting, where $K > 1$. A voter sends its history information as bit map, with each bit position corresponding to a voting round and indicating whether a YES or NO vote was intended by this voter in the corresponding round. Upon collecting histories from enough number of voters such that a majority determination of the actual votes can be ascertained, B compares its earlier decision for this round. If

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

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-109-0

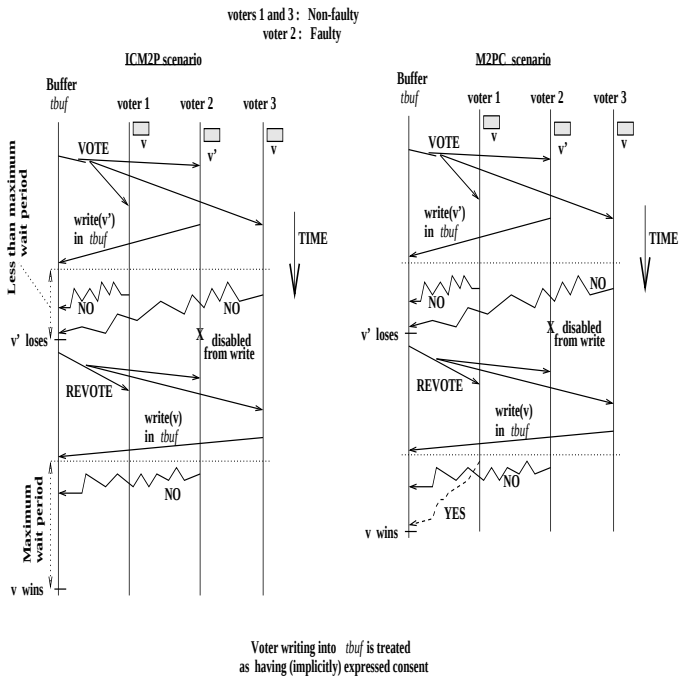

Fig. 3. A voting scenario with M2PC and IC-M2PC protocols

the two decisions match, the result is delivered to the user; otherwise, the result is discarded.

Consider a sample scenario shown in Figure 4. After round 4 in IC-M2PC mode, history vectors are collected, whereupon B finds that the decision in round 2 was erroneously made, and hence caused a discard of the result d_2. Since only 1 out of the 4 decisions was erroneous, B attributes the problem as arising from benign message loss conditions. At the end of round 6 however, B finds that both the decisions were erroneously made, and hence suspects a persistent message loss. This causes B to switch to the safer mode: M2PC. That B receives the consent/dissent messages without much loss in rounds 7-9 causes a switch back to the optimal mode: IC-M2PC, in round 10.

We introduce a *resilience* parameter M, which is the number of implicit YES votes needed in favor of a data d' (in addition to the YES vote associated with the proposer of d') for B to accept d' as good, where $f_m \leq M \leq (N - f_m - 1)$. Thus[2], if L NO votes are received by B from the remaining $(N-1)$ voters during the dissent timeout period T_w, the condition:

$$(N - 1 - L) \geq M \quad (1)$$

determines if the acceptance threshold has been met to declare the data d' as good. M is a tunable parameter in the range $[f_m, N - f_m - 1]$ to control the data miss rate ζ. Choosing a value of M much higher than f_m reduces the occurrence of false negatives, i.e., lowers ζ.

Our voting protocol thus embodies 2 modes of operation: i) the light-weight IC-M2PC that performs well under normal cases when packet loss is small and failures are benign, and ii) the heavy-weight M2PC that is more resilient against severe failures but incurs more message overhead. The protocol evaluates these tradeoffs at run-time, and then switches to the appropriate mode based on the sensed failure conditions.

[2]In M2PC, there is no need for the M parameter, i.e., f_m YES votes (besides the YES from the proposer of d') suffice to accept d' as good.

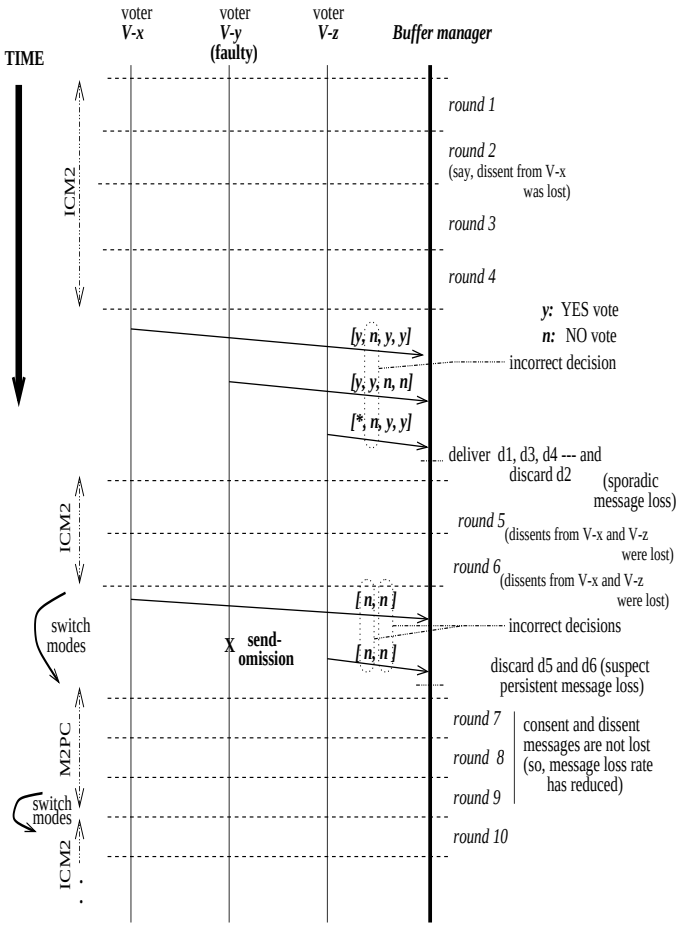

Fig. 4. Vote history to check commit decisions ('*': voter V_z is unaware of round 1)

D. Voting mode switching control

Since the network message loss rate l may not be directly available to the voting protocol layer, l is determined based on the data misses observed at the interface to the user.

The message loss rate l impacts the data miss rate ζ in different ways in M2PC and in IC-M2PC. A higher l causes an increase in the duration of data proposal phase due to retransmissions of a candidate data until B and at least $\frac{N}{2}$ voters receive the data. The non-receipt of a candidate data at one or more voters causes them to cast a NO vote for the data (for safety reasons), which makes it possible that no data is delivered to the end-user even after $2f_m + 1$ iterations. Furthermore, the time-to-complete a voting iteration (i.e., decide on the abort or commit of a candidate data) increases in M2PC due to retransmissions of the lost votes for a candidate data. The resulting increase in the time-to-deliver a data may cause more frequent violations of the timeliness

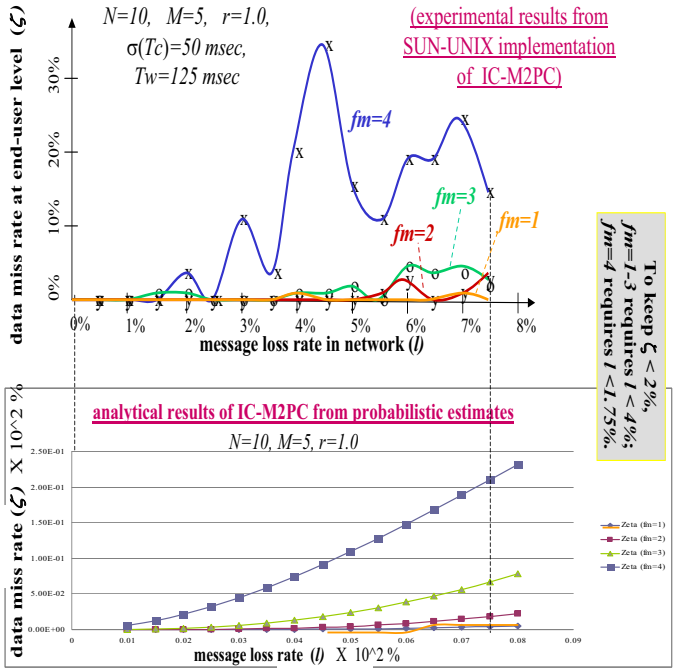

Fig. 5. Experimental and analytical results on data miss in IC-M2PC mode

constraint Δ. In IC-M2PC however, the lost NO votes within a waiting period T_w are treated as consents, leading to a premature tentative commit in favor of a faulty candidate data that subsequently gets invalidated during the history-vector based check. Regardless of how the message-loss phenomenon inter-plays with the data delivery mechanisms in M2PC and IC-M2PC, an increase in l causes an increase in ζ.

The voting layer employs a monitor at the data delivery point to the user to determine the missing data, if any. For this purpose, the monitor agent at B affixes sequence numbers to the voting rounds generated during a IC-M2PC run. At the point of delivery to the end-user, the agent at the user delivery point checks for missing sequence numbers. The observed data miss rate (ζ) allows the monitor to estimate l therefrom.

For the M2PC protocol, the monitor agent at the user end keeps track of the number of missing data in the various voting rounds (using the monitor-assigned sequence numbers). This information, combined with the observed number of missing votes (from the required N votes) in each round during M2PC execution, allows estimating of the message loss rate l.

Figure 5 provides the experimental results from IC-M2PC implementation on SUN-UNIX system for the case of [$N = 10, M = 5, r = 1.0$], with $f_m = 1, 2, 3, 4$ and $l = 0.5 - 8\%$. For $f_m = 4$, ζ increases to beyond 4% for $l \approx 3\%$. For $f_m = 3$, ζ increases to beyond 3.5% for $l \approx 5\%$. For $f_m = 2$ and $f_m = 1$, $\zeta < 1.5\%$ for $l < 4\%$. These results, combined with the information on voting message overhead and latency, allow the designer to set an acceptable operating region for IC-M2PC vis-a-vis M2PC.

We now describe a case study of replicated web services where M2PC and IC-M2PC protocols can be employed to infuse fault-tolerance during query processing.

IV. Case Study of Replicated Web-Based Information Service

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 [11]. 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). 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 [12], [13] (e.g., sensors deployed in a field). 'subscriber' clients query the repository to identify objects of interest via read operations on the data servers. 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 web service is more vulnerable to attacks than the back-end data servers that host the web service. This is because of the security weaknesses and diverse usage profiles inherent in the web service operations. Servlet replication is often employed as the basic means to counter the effects of attacks and enhance the service availability. Our goal here is to examine the web service reliability in terms of our probabilistic models in a scenario of servlet replication.

A. Servlet replication support mechanisms

The servlet replicas are capable of independently processing the queries and computing the results returned to the client. The parameter f_m, which depicts the maximum number of replicas that the system designer assumes as fault-prone, strongly impacts the choice of N: the total number of replicas. 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. See Figure 6.

The degree of replication N is determined by the cost of replication weighed against the need for a higher reliability of the web service. For a given N, the network message-loss phenomenon however impacts the M2PC and IC-M2PC operations in different ways, as measured in terms of voting message overhead and latency.

Voting latency directly impacts the user-level data miss rate ζ — and hence the query success rate.

B. Web service reliability: IC-M2PC vs M2PC

The performance measure ζ is an indicator of *service availability*, depicting how often a client query fails to return a correct result. Here, any attempt to return a faulty result (i.e., a corrupted result or a result that is delayed longer than Δ) is detected by the replica voting protocol.

The candidate output result of a query from a servlet can be decided as correct upon receiving $f_m + 1$ consent votes for

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-109-0

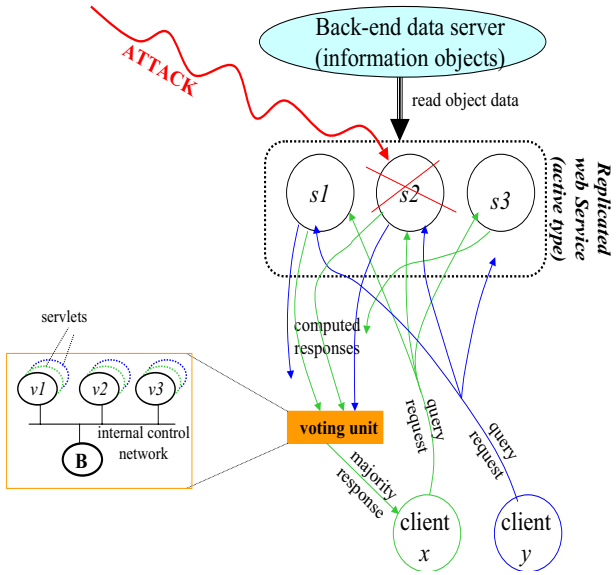

Fig. 6. Servlet replication technqiues for query processing

the result, under the condition: $f \leq f_m$, where f is the actual number of attacked servlets and $1 \leq f_m < \lceil \frac{N}{2} \rceil$ [15].

As for the timely return of a result, the higher latency in voting with IC-M2PC may increase ζ (and hence lower the content availability), in comparison with M2PC. IC-M2PC however incurs less message overhead (and hence consumes less bandwidth) than M2PC. Furthermore, the protocol designer may conservatively set $f_m = \lceil \frac{N}{2} \rceil - 1$ for a given N (for an increased margin of consent voting) — even though the knowledge about f may instil confidence in the designer to set f_m optimistically to a lower value.

The resilience of a web query is determined by the ability of the web service layer to reconfigure the query processing without exceeding the latency limit Δ when the servlet currently returning the query result is deemed to have failed. A reconfiguration involves aborting a query result due to lack of enough consent votes and soliciting a new result for the query from a different servlet (i.e., moving to a next iteration in the current round). Since IC-M2PC infers the consent votes for a query result only by implicit means: namely, the non-receipt of dissent votes, a message loss in the network may cause IC-M2PC to initially proceed with incorrect results which then get discarded later by the history-vector mechanism (even if the time elapsed is less than Δ). So, IC-M2PC may incur a higher ζ (and hence entail a lower resilience) than M2PC when l and/or f is large.

V. Conclusions

We considered the development of message-optimal voting protocols for data collection in sensor networks. The environment is one of a lossy data transport network, compounded by malicious failures of data collection devices. The functional extensions needed to the basic form of 2-phase majority voting are the provisioning of additional protection layers to sanitize the voting decisions in the presence of sustained message loss. We described two modified versions of 2-phase commit: M2PC and IC-M2PC. M2PC is based on explicit exchange of YES and NO votes; IC-M2PC attempts to reduce the message overhead by suppressing the YES votes. The required augmentations to the protcols to increase their robustness and performance were also described, along with experimental data from a prototype voting system.

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 sensing system. A main thrust is to reduce the message overhead, while keeping the voting latency to within tolerable limits. Our protocol is highly adaptive to the various types of failures occurring in the data collection environment (such as sensor networks), to meet the goal. We also described a case study of replicated web service employing the voting protocols to provide a fault-tolerant query processing.

ACKNOWLEDGEMENT

The paper has been approved by AFRL for Public Release (Distribution Unlimited): 88ABW-2010-3936 dated 21 Jul 10.

References

[1] H. Kopetz and P. Verissmo. 'Real Time Dependability Concepts'. Chap.16, *Distributed Systems*, S. Mullender, Addison-Wesl. Co., 1993.

[2] O. Babaoglu. 'Non-blocking Commit Protocols'. Chap. 7, *Distributed Systems*, ed. S. Mullender, Addison-Wesley Publ. Co., 1993.

[3] B. Hardekopf, K. A. Kwiat, and S. Upadhyaya. 'Secure and Fault-tolerant Voting in Distributed Systems'. In proc. IEEE *Aerospace Conference*, Big Sky (MT), pp. 1117-1126, March 2001.

[4] R. R. Brooks and S. Iyengar. Chap. on 'Sensor Fusion and Approximate agreement'. In *Multisensor Data Fusion*, Prentice-Hall Publ., 1998.

[5] R. D. Prisco, B. Lampson, and N. Lynch. 'Fundamental study: Revisiting the Paxos algorithm'. In *Theoretical Computer Science*, 243:35–91, 2000.

[6] P, Jalote and et al. 'Atomic Actions on Decentralized data'. Chap. 6, *Fault-tolerant Systems*, John-Wiley Publ. Co., 1995.

[7] W. Du, J. Deng, Y.S. Han, and P.K. Varshney. 'A Witness-based Approach for Data Fusion Assurance in Wireless Sensor Networks'. In proc. IEEE-GLOBECOM'03, pp. 1435-1439, Dec.2003.

[8] S. Forrest, A. Somayaji, and D.H. Ackley. 'Building Diverse Computer Systems'. In proc. 6th Workshop on *Hot Topics in Operating Systems* (HotOS-VI), IEEE, ISBN: 0-8186-7834-8, 1997.

[9] M. Castro and B. Liskov. 'Practical Byzantine Fault Tolerance'. In proc. 3rd Symp. on *Operating Systems Design and Implementation*, New Orleans (LA), ACM-DL, ISBN:1-880446-39-1, Febr. 1999.

[10] M. A. Malek, G. Ganger, G. Goodson, M. Reiter and J. Wylie. 'Fault-scalable Byzantine Fault-tolerant Services'. In proc. 20th ACM symp. on *Op. Sys. Principles*, ACM SIGOPS, Brighton (UK), ISBN:1-59593-079-5, Oct. 2005.

[11] S. Ghandeharizadeh, C. Papadopoulos, M. Cai, R. Zhou, and P. Pol. 'NAM: A Network Adaptable Middleware to Enhance Response Time of Web Services'. Chap. II, *Web Services: Researches and Chalenges*, ed. L. J. Zhang, Cybertech Publishing, 2008.

[12] S. Pollack and W. K. McQuay. 'Joint Battlespace Infosphere Applications Using Collaborative Enterprise Environment Technology'. In *proc. SPIE-5820, Defense Transformation and Network Centric Systems*, Ed. Raja Suresh, Bellingham (WA), pp. 235-242, 2005.

[13] IBM Software Group. 'IBM Infosphere Master Data Management Server: Technical Overview'. *White Paper*, Feb. 2008.

[14] A. Sabbir, K. A. Kwiat, K. Ravindran, and M. Rabby. 'Do Centralized and Distributed Voting Methods Complement Each Other ?? A Case Study of Replica Voting Protocol Design'. in *Technical Report*, Dept. of Computer Science, The City University of New York, Sept. 2009.

[15] K. A. Kwiat, K. Ravindran, C. Liu, and A. Sabbir. 'Performance and Correctness Issues in Secure Voting for Distributed Sensor Systems'. in proc. *SPECTS'02*, San Diego (CA), July 2002.

Copyright (c) IARIA, 2010 ISBN: 978-1-61208-109-0