Meaningful Metrics for Evaluating Eventual Consistency*

João Barreto and Paulo Ferreira

INESC-ID/Technical University Lisbon,
Rua Alves Redol, 9, 1000 Lisboa, Portugal
{joao.barreto,paulo.ferreira}@inesc-id.pt

Abstract. Optimistic replication is a fundamental technique for supporting collaborative work practices in mobile environments. However, eventual consistency, in contrast to immediate strong consistency in pessimistic replication, is much harder to evaluate. This paper analyzes different metrics for measuring the effectiveness of eventually consistent systems. Using results from a simulated environment of relevant optimistic replication protocols, we show that each metric hides previously undocumented side effects. These add considerable imprecision to any evaluation that exclusively relies on a single metric. Hence, we advocate a combined methodology comprising three complementary metrics: commit ratio, average agreement delay and average commit delay.

1 Introduction

The emerging paradigms of ubiquitous and pervasive computing [1,2] enable a wide range of promising collaborative applications and systems, asynchronous groupware applications [3,4] such as shared document editing, cooperative engineering or software development [5,6,7], collaborative wikis [8,9], shared project and time management [10,11], distributed file [12,13] or database systems [14].

It is well known that an easy way to support such applications and systems is by data replication [15]. Optimistic replication (OR) [15] is especially interesting in ubiquitous and pervasive environments, due to their inherent weak connectivity. OR, in contrast to traditional pessimistic replication, enables access to a replica without *a priori* synchronization with the other replicas. Hence, it trades conflict-free, strong consistency for the possibility to access shared data anytime and anywhere. As collaboration in ubiquitous and pervasive computing environments becomes popular, the importance of OR increases.

Inevitably, however, consistency in OR is challenging [15]. Since one may update a replica at any time and under any circumstance, the work of some user or application at some replica may conflict with concurrent work at other replicas. Hence, instead of immediate consistency, OR offers eventual consistency. A *replication protocol* is responsible for disseminating updates among replicas

* This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and FCT Project Byzantium (PTDC/EIA/74325/2006).

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part II, LNCS 6272, pp. 503–515, 2010.
© Springer-Verlag Berlin Heidelberg 2010

and eventually scheduling them consistently at each of them, according to some consistency criterion. The last step of such a process is called *update commitment*. Possibly, it may involve aborting (and rolling back) updates, in order to resolve conflicts with updates that have committed.

In this paper we are concerned with correctly evaluating the usefulness of OR systems. In pessimistic replication, we can tell how useful a system is by measuring (i) the average proportion of time during which a given client was able to access some replicated object, (ii) access latency, and (iii) memory and network overheads. However, in OR, component (i) is no longer meaningful, as the availability of an OR system is 100%[1]. Instead, there is a new component to be measured: how good does the system converge in background towards eventual consistency? Answering such a question is inherently more complex than in the case of pessimistic strong consistency, as it depends one two contending dimensions:

- Firstly, how much time does it take for each update to be (eventually) committed (or aborted, in the worst case)?
- Secondly, how many updates were aborted due to unresolved conflicts?

OR system should minimize the first and maximize the second. In other words, minimize the time window during which users and application are forced to access possibly inconsistent data, at the expense of as least aborted work as possible. To a great extent, the ability of the OR system to do that determines whether the users and applications working in weakly connected environments such as ubiquitous or pervasive computing are willing to rely on such a highly available, yet weakly consistent, alternative. If not, they will easily resort to more limitative, yet less risky, pessimistic replication solutions.

Perhaps surprisingly, most OR literature seems to neglect the importance of correctly and thoroughly evaluating eventual consistency. We observe that most considered metrics are either not related, or only very indirectly related to the ability of the systems to achieve eventual consistency. Furthermore, common metrics related to eventual consistency, such as commit ratio, agreement and commitment delays are typically considered individually. This paper questions the meaningfulness of such metrics in evaluating the above two dimensions.

From a thorough experiment with three relevant OR protocols in a simulated mobile collaborative environment, we show that each such metric is unexpectedly affected by subtle, yet considerably misleading side effects. In most scenarios, exclusively relying on a single metric to compare different OR systems can lead to a unexpectedly imprecise and incomplete conclusions.

Our main contribution is, therefore, a methodology for precise and complete evaluation of OR systems. We advocate that correct evaluation of OR systems should rely on a combined analysis of three main metrics, namely commit ratio, average agreement delay and average commit delay, and not less that those; and

[1] Although this paper focuses on pure, full availability OR, our results are also extensible to eventually consistent solutions that can have less than 100% availability, such as TACT [16], VFC [17] or Bayou's Session Guarantees [18].

describe under which experimental conditions each metric should be measured. We support such claims with our experimental results and analysis.

The remainder of the paper is organized as follows. Section 2 describes how literature addressing related work on eventually consistent OR systems evaluates the corresponding solutions. Section 3 describes the experimental model and evaluated protocols that provided us with illustrative results that support our analysis. Section 4 then addresses three metrics for evaluating eventual consistency, applies each such metric to a simulation-based experiment and exposing side effects and imprecisions of each metric. Finally, Section 5 draws conclusions, advocating a methodology and guidelines for correct evaluation of eventually consistent systems.

2 Related Work

The road to eventual consistency has two main stages: first, individual replicas schedule new updates that they learn of in some way that is safe, i.e. free of conflicts. Second, each such tentative schedule is submitted as a candidate for some a distributed commitment protocol, which will, from such an input, agree on a common schedule which will then be imposed to every replica. Commitment is, thus, the key step that pushes the OR system towards eventual consistency [15]. One may distinguish four main commitment approaches in OR literature.

A first approach may be designated as the unconscious approach [19]. In this case, the protocol ensures eventual consistency; however, applications may not determine whether replicated data results from tentative or committed updates. These systems are adequate for applications with very weak consistency demands, for which it is neither relevant to know whether previous tentative is committed or aborted, nor when such decision has been taken. For example, Usenet, DNS and Roam [20].

Other approaches, however, allow explicit commitment. A second approach for commitment is to have a replica commit an update as soon as the replica knows that *every other replica* have received the update [21,22]. This approach has the drawbacks of halting while any single replica becomes unavailable and of not handling update conflicts.

A third approach is a primary commit protocol [23]. It centralizes commitment into a single distinguished primary replica that establishes a commitment total order over the updates it receives. Such an order is then propagated back to the remaining replicas. Primary commit is able to rapidly commit updates, since it suffices for an update to be received by the primary replica to become committed. However, should the primary replica become unavailable, commitment of updates generated by replicas other than the primary halts.

Finally, a fourth approach is by means of voting [24,25,26]. Here, divergent update schedules constitute candidates in an election, while replicas act as voters. Once an update schedule has collected votes from a quorum of voters that guarantee the election of the corresponding candidate, its updates may be committed in the corresponding order. Voting eliminates the single point of failure of primary commit.

In particular, Keleher introduced voting in the context of epidemic commitment protocols [27]; his protocol is used in the Deno [28] system. The epidemic nature of the protocol allows updates to commit even when a quorum of replicas is not simultaneously connected. Each replica, upon learning of enough votes that guarantee that a given candidate update has already collected a plurality of votes, unilaterally decides to commit the elected candidate. The same replica considers the remaining rival updates as aborted. Similarly to *Primary*, such commitment and abort decisions then propagate epidemically to other replicas. This means that a replica can commit or abort an update by one of two means: either (1) by receiving enough votes, or (2) by epidemically hearing about the decision that some other replica that has decided by (1). As an update is elected in a given election, a new election starts.

Deno requires one entire election round to be completed in order to commit each single update [29]. The same happens with a closely related approach proposed by Holliday et al. [30], which uses traditional coteries, such as majority (instead of plurality).

VVWV [31] is an improvement over Deno and Holliday et al.'s protocols that is able to commit chains of multiple happens-before-related updates on a single election round. This is a frequent occurrence in OR systems running on frequently partitioned environments, where applications will often issue sequences of causally related updates before the first update in the sequence even gets committed. In this case, the sequence can be committed as a whole in a single election, rather than requiring a series of elections (one per each update in the sequence).

Experimental evaluation of proposed commitment protocols is often scarce and incomplete. Ideally, it should answer the two questions mentioned in Section 1, which are the main concerns of applications and users of the OR system. Most work proposing OR systems limits their evaluation to memory and time overheads associated with the maintenance of tentative versions and replica-to-replica synchronization sessions (e.g. Coda [32], Bayou [23], ROAM [20]). Others do focus on the ability to ensure eventual consistency, but analyze only a subset of the metrics that we discuss next, namely commit ratio and average commitment delay. Examples include Deno [27,28] and VVWV [31]. As we show next, such an analysis is incomplete as it conceals the noise of some side effects.

3 Experimental Methodology

We have implemented a simple OR system with three variants, each using a different commitment protocol offering explicit eventual consistency, from those described in Section 2: *Primary*, *Deno* and *VVWV*. For simplicity, every implementation uses a syntactic approach to schedule updates and detect conflicts. Update schedules consist of ordered sequences of updates related by the happens-beforerelation [33]. If some replica receives an update that is concurrent (by happens-before) with the most recent update in its local update schedule, the former update is conservatively considered as conflicting with the latter; hence one of them will (eventually) commit and the other abort.

We ran *C#* implementations of *Primary*, *Basic WV* and *VVWV* side-by-side in a simulated environment. The simulator includes a collection of replicas of a common logical object, randomly distributed by a set of network partitions. Time is divided into logical time slices; at each time slice, each replica (1) with a given *mobility probability*, migrates to a different, randomly chosen, network partition; (2) executes a one-way synchronization session from some partner, randomly selected from the set replicas present in its current partition; and, (3) generates, with a given *update probability*, one tentative update.

All the experiments discussed herein use a system of five replicas. We found such a dimension to be sufficiently complete to illustrate our findings regarding each metric that we address in the next sections. We consider two alternative scenarios: low and high mobility, with mobility probabilities of 20% and 40%, respectively. We adopt the following procedure for obtaining the measurements on which our evaluation is based. We consider samples comprising measurements from five simulation runs. Each run lasts for 2000 logical time steps and, before each run, we initialize the simulator with a distinct randomization seed, so that each of the five runs is effectively distinct. The results we present next are an average of the measurements taken from the five runs.

We should add that this particular choice of protocols, mobility model and update model was made with the sole intention of illustrating, in the most simplified manner, enough experimental evidence that supports the conclusions we draw in the following analysis. Needless to say, a larger set of state-of-the-art eventually consistent systems exists, as Section 2). Moreover, more realistic mobility models (e.g. [34]) and update models can be considered (e.g. [31]). The conclusions we draw next, based on this simplified model and set of solutions, are also nonetheless valid with other systems, mobility models and update models.

4 Evaluation

4.1 Commit Ratio

Probably the most common metric for evaluating the ability of an OR system to achieve eventual consistency is commit ratio. We define commit ratio as *the percentage of issued updates that each protocol commits as executed at all replicas* (as opposed to the updates that it aborts). Figure 1 compares the commit ratios of each protocol. As in previous measures, we consider both low and high mobility, for and increasing number of partitions.

Commit ratio is directly affected by the efficiency of each evaluated protocol, because if updates remain in their tentative state for longer periods, the probability of aborts is higher. What some authors call *hidden conflicts* [35] explains this. Essentially, a hidden conflict occurs when a replica is aware of two or more concurrent tentative versions. While no commitment decision has been taken about which of such tentative versions will commit (and which will abort), the replica must choose one of them to present to its applications. Any further updates that the local applications may issue in the meantime will causally depend on the chosen tentative version, and will be concurrent (hence conflicting) with

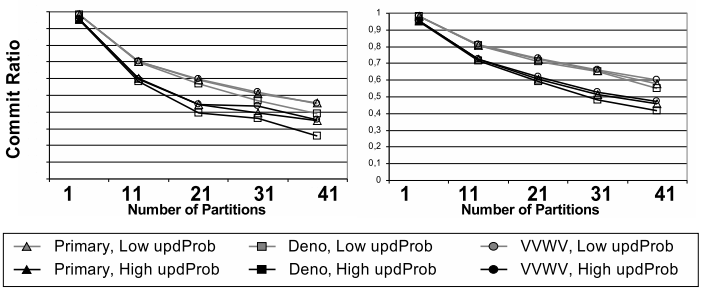

Fig. 1. Commit ratios vs. number of partitions. Both low and high site mobility are considered (resp., *mobProb*=25% and *mobProb*=75%); and both low and high update probabilities are considered (resp., *updProb*=$\frac{0.5\%}{5}$ and *updProb*=$\frac{0.9\%}{5}$).

the other (hidden) tentative versions. If, by chance, the commitment protocol decides that one of those concurrent versions is the one to commit, then all the updates issued by the local applications in the meantime will be regarded as conflicting with the new committed version, hence aborted. We say these aborted updates were caused by a hidden conflict.

Clearly, the probability of hidden conflicts increases with the time it takes for the commitment decision to be taken. Hence, lower commit ratios reflect longer delays imposed by the update commitment process. Figure 1 shows that, as expected, Primary and VVWV ensure higher ratios than Deno, especially as partitioning grows and/or mobility decreases; i.e., as connectivity becomes weaker. In contrast, Primary and VVWV have nearly similar ratios. If one relied solely on this metric, one would conclude that both protocols are identically efficient in achieving eventual consistency. However, this metric hides a crucial side of the actual experience of the applications and users of the OR system.

In fact, many applications using an OR system will tolerate accessing a weakly consistent tentative value for some operations, but will wait until their tentative work is finally incorporated into the strongly consistent value before proceeding with some critical operations. The delay separating both moments is, evidently, a crucial aspect to the effectiveness of a commitment protocol.

Of course, it is important that the commitment decision taken by the OR system does not entail loosing any tentative work that application or user has been carrying out. Still, in many scenarios it is just as important to learn what that decision is within a reasonable time window. In most cases, a high delay before receiving that decision can be as bad as knowing that our recent tentative work needs to be discarded. Hence, we need to also take into account metrics that more precisely reflect the delays imposed to users and applications.

4.2 Average Agreement Delays (AAD)

One way to take delays concerning eventual consistency into account is to measure the agreement delay of a commitment protocol. This is a combined metric, which reflects the protocol's commitment and abort delays.

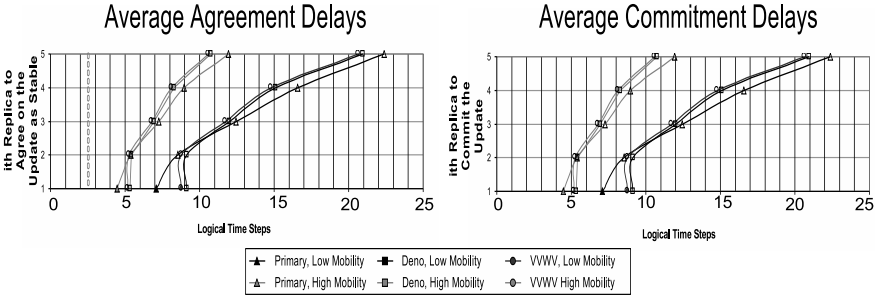

Fig. 2. Average agreement delays, for low site mobility (*mobProb*=25%) and high site mobility (*mobProb*=75%), *updProb*=$\frac{0.5\%}{5}$ = 0.1% and 11 Partitions

We define commitment delay for an update, u, at some replica, r, as *the number of time steps separating the step where some replica issued u and the step where r committed u*. When u aborts, rather than committing, we reason about its abort delay. We define the abort delay of u at r as *the number of logical time steps separating the issue of u and the first step where r commits a concurrent update (by happens-before) with u*. Note that this definition does not require r to actually receive u and then abort it; it is sufficiently general to also consider the cases where u does not even propagate to r, because r has already (implicitly) aborted u by committing a concurrent update. Finally, the agreement delay of an update is either its commitment and abort delay, depending on whether the update commits or aborts.

A second experiment studies the average agreement delays (AAD) of *Primary*, *Deno* and *VVWV* under different networking conditions, both in terms of number of partitions and of mobility probabilities. Figure 2 (left-hand graph) presents the results, distinguishing two mobility scenarios (low mobility, with *mobProb*=25% and high mobility, with *mobProb*=75%) for a situation of 11 partitions. For space limitations, we omit the figures for different partition numbers.

Analisys of AAD. As expectable, AADs increase significantly as we weaken connectivity with more partitions. This increase may be of almost one order of magnitude as we depart from a single-partition situation to 11 partitions with low mobility. High mobility of replicas strongly reduces the effect of partitioning, typically accelerating agreement to less than half the delay with low mobility.

We start by comparing the AADs of Primary and VVWV (we proceed shortly to compare VVWV with Deno). From Figure 2 (left-hand graph), we see that *Primary* generally performs substantially better than *VVWV* for the first few replicas.

For a higher replica order, however, the AADs of *Primary* converge with and, in some cases, become higher than the AADs of the weighted voting alternatives (VVWV and Deno). This behavior becomes more evident with larger systems.

This observation has already been documented in the context of the Deno replicated system [28]. Keleher et al. explain it by the fact that, although the

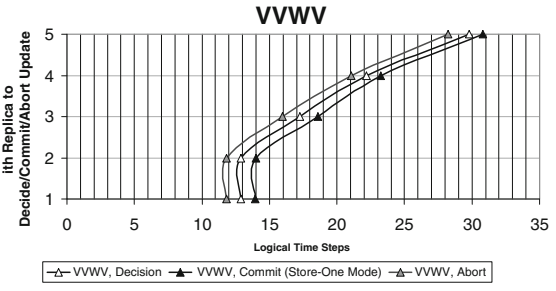

Fig. 3. Average decision vs. commitment vs. abort delays for VVWV in a high-contention scenario. 41 Partitions, *mobProb*=25%, *updProb*=$\frac{0.9\%}{5}$ = 0.18%.

Primary is able to commit rapidly (since it suffices for the update to arrive at one replica, the primary), such a decision propagates relatively slowly to other replicas. This is because all other replicas must learn of the commitment, directly or indirectly, from the primary replica. In contrast, *Deno* and *VVWV* enable distinct replicas to either learn the decision from other replicas (as in the case of *Primary*), or decide the update independently by receiving sufficient votes.

Of course, an important aspect is to compare both epidemic weighted voting protocols. As expectable, the AADs of *VVWV* become increasingly lower than those of *Deno* as connectivity degrades with partitioning and/or low mobility.

Considerations on AAD. AAD gives a realistic picture of the user experience that a commitment protocol provides, since it tells us how much time a user has to wait before receiving either a commit or an abort decision on some tentative piece of work. However, we argue that AAD is not sufficiently meaningful as a measure of the effectiveness of a protocol. We support such an argument by decomposing AAD into commitment and abort delays, which allows for a better understanding of the actual meaning of the AAD measure.

Figure 3 distinguishes the commitment and abort delays (as well as the compound, agreement delays) of VVWV in a particular setting, with 41 Partitions and *mobProb*=25%. We chose a relatively high update probability (*updProb*=0.18%), so that it induced a high contention scenario where comparable numbers of commits and aborts occurred. As the results show, the time necessary to commit an update is markedly higher than to abort an update. In the setting that Figure 3 considers, commitment delay in VVWV is 18% and 9% higher than abort delay for the first replica and last replica (to either commit or abort), respectively. Significant differences exist with Primary (25% and 12%) and Deno (31% and 15%) too. We obtained similar conclusions for the other protocols, which we omit for space limitations. Moreover, other values of *mobProb*, *updProb*, and *numPart* also yield relevant differences between commitment and abort delays.

Intuitively, we may justify the above difference by the fact that committing an update requires a complete election to initiate (by proposing the update as a

candidate) and complete (by having the system vote for the update and electing it); whereas aborting does not. In fact, aborting a given update happens once some update that is concurrent with the former commits; the election of the latter update may already have started when the aborted update was issued. As an example, consider that an update u_1, issued at time step 10, has already collected a majority of votes, but a given replica, r, has neither yet received the update nor its voting information. Assume that, at time step 21, r issues update u_2, concurrently with u_1. Then, at time step 22, r receives u_1 and its voting information. Hence, at time 22, r commits u_1, which has a commitment delay of $22 - 10 = 12$ at r; and, consequently, r aborts u_2, which has an abort delay of $22 - 21 = 1$ at r.

Therefore, the AAD measure tends to benefit (undesirable) protocols that may be slow to commit but abort frequently. When compared to a (more interesting) protocol that commits faster and more frequently, the former protocol may achieve AADs that are only slightly higher or possibly even lower than those of the latter protocol. This is due to the deceiving impact of *small delays of frequent aborts*, which replace longer-to-complete commits. Further, abort delays have a considerably high variance, due to their arbitrary nature, as described above. Hence, when comparing two protocols that commit similar number of updates, abort delay contributes with arbitrary noise to the overall AADs.

The previous observations show that the meaningfulness of AAD is actually very limited. This might suggest that a more interesting alternative would be to consider commitment delay only, which we discuss next.

4.3 Average Commitment Delays (ACD)

Figure 2 (right-hand graph) presents average commitment delays (ACD) for each protocol, with 11 partitions, in two different scenarios of replica mobility. We omit the figures for other partition numbers, again for space limitations.

Analysis of ACD. Every observation that we make above for AAD remains valid in the case of ACD. Namely:

- ACD grows as partitions increase and as mobility decreases.
- Primary obtains lower ACDs for the first replicas, while the weighted voting protocols outperform Primary for higher order replicas to commit.
- VVWV attains lower ACDs than Deno, with more partitions.

The essential difference between ACD and AAD, as we discussed in Section 4.2, is that ACD is immune to the noise that abort delays introduce in AAD. Since ACD does not take abort delays into account, it is generally higher than AAD, as expectable.

Figure 2 (right-hand graph) unveils the relative variation in ACD from Primary to VVWV, and from VVWV to Deno, respectively. Curiously, ACDs reveal that the noise of abort delays on AAD always benefits Primary over VVWV. In fact, the difference between the ACDs of Primary and VVWV decreases in comparison to the corresponding difference in terms of AAD. For instance, to the

highest difference between the AADs of VVWV and Primary, 152.6% (which occurs with 41 partitions, low mobility), corresponds an ACD of 113.5%. We verify the inverse when concerning Deno: the difference between the ACDs of VVWV and Deno increases substantially in comparison to the corresponding difference in terms of AAD. As an example, in the previous setting, the AAD of Deno is 16.3% higher than VVWV's, while the difference in ACD is of 31.4%. Summing up, the evaluation of the protocols from the point of view of ACD, rather than of AAD, is substantially more positive for VVWV.

We may further characterize each protocol by individually comparing the evolution from its AADs to its ACDs. Interestingly, in general (with some exceptions), Deno has the highest gap between AACs and ACDs. Since aborts tend to contribute to lowering AAD (in relation to ACD), such an observation suggests that Deno has lower commit ratios than the other protocols. Comparing VVWV with Primary, a similar phenomenon happens for a number of cases; accordingly, this suggests a lower commit ratio of VVWV in relation to Primary.

However, recalling the arbitrarity of the noise of abort delays in AAD, the above previsions are very limited in accuracy. The next section tries to validate them by studying the effective commit ratios of each protocol.

Considerations on ACD. ACD gives a better understanding of the efficiency of a protocol, since it avoids the noise that abort delays introduce in AAD. However, it is still not a completely meaningful measure, due to the effect of hidden conflicts, as we explain now.

Aborts due to hidden conflicts are prone to occur since commitment is not immediate. By using a protocol that effectively commits faster, one reduces the probability of hidden conflicts; hence, on average, one aborts less updates. Therefore, the number of discarded updates also characterizes the effective efficiency of a protocol.

However, ACD does not take discarded updates into account. It is a useful measure to compare the efficiency of two protocols in executions where both commit a similar (or nearly similar) set of updates. However, when one protocol discards more updates because it is not as fast as another one, ACD does not necessarily expose such a difference. It is, thus, necessary to complement this metric with the commit ratio metric to have a complete understanding of the each protocol's efficiency in achieving eventual consistency.

5 Conclusions

The ability of an OR system to rapidly achieve eventual consistency at the expense of minimal aborted work is a crucial factor for the usefulness of that OR system. To a great extent, it determines whether the users and applications working in weakly connected environments such as ubiquitous or pervasive computing are willing to rely on such a highly available, yet weakly consistent, alternative; or, otherwise, they will resort to a more limitative, yet less risky, pessimistic replication solution.

Perhaps surprisingly, most OR literature seems to neglect the importance of correctly and thoroughly evaluating eventual consistency. We observe that most considered metrics are either not related, or only very indirectly related to the ability of the systems to achieve eventual consistency. Furthermore, common metrics related to eventual consistency, such as commit ratio, agreement and commitment delays are typically considered individually.

In this paper, through a simple experiment with three OR commitment protocols, we show that such three metrics are affected by important obscure side effects, which, to the best of our knowledge, were previously undocumented. Consequently, each metric is not sufficiently meaningful by itself. We therefore propose a combined methodology for evaluating eventual consistency, which uses the three metrics to evaluate different aspects of eventual consistency. The present paper is an example of an application of such a methodology to compare three relevant commitment protocols: primary, Deno and VVWV.

References

1. Weiser, M.: The computer for the twenty-first century. Scientific American 265, 94–104 (1991)
2. Forman, G.H., Zahorjan, J.: The challenges of mobile computing. Computer 27, 38–47 (1994)
3. Wilson, P.: Computer Supported Cooperative Work: An Introduction. Oxford, Intellect Books (1991)
4. Carstensen, P.H., Schmidt, K.: Computer supported cooperative work: New challenges to systems design. In: Itoh, K. (ed.) Handbook of Human Factors, pp. 619–636. Asakura Publishing (1999) (in Japanese), English Version available from, `http://www.itu.dk/people/schmidt/publ.html`
5. Cederqvist, P., et al.: Version management with CVS [Online Manual] (1993), `http://www.cvshome.org/docs/manual/` (03.09.2002)
6. Pilato michael, C., Collins-Sussman, B., Fitzpatrick, B.W.: Version Control with Subversion. O'Reilly, Sebastopol (2004)
7. Chou, Y.: Get into the Groove: Solutions for Secure and Dynamic Collaboration (2006), `http://technet.microsoft.com/en-us/magazine/cc160900.aspx`
8. Leuf, B., Cunningham, W.: The wiki way: Quick collaboration on the web. Addison-Wesley, Reading (2001)
9. Ignat, C.L., Oster, G., Molli, P., Cart, M., Ferrié, J., Kermarrec, A.M., Sutra, P., Shapiro, M., Benmouffok, L., Busca, J.M., Guerraoui, R.: A comparison of optimistic approaches to collaborative editing of wiki pages. In: CollaborateCom, pp. 474–483. IEEE, Los Alamitos (2007)
10. Leonard Kawell, J., Beckhardt, S., Halvorsen, T., Ozzie, R., Greif, I.: Replicated document management in a group communication system. In: CSCW 1988: Proceedings of the 1988 ACM conference on Computer-supported cooperative work, p. 395. ACM Press, New York (1988)
11. Byrne, R.: Building Applications with Microsoft Outlook 2000 Technical Reference. Microsoft Press, Redmond (1999)
12. Nowicki, B.: Nfs: Network file system protocol specification. Internet Request for Comment RFC 1094, Internet Engineering Task Force (1989)

13. Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S., Smith, F.D.: Andrew: a distributed personal computing environment. Communications of the ACM 29, 184–201 (1986)
14. Thomas, G., Thompson, G.R., Chung, C.W., Barkmeyer, E., Carter, F., Templeton, M., Fox, S., Hartman, B.: Heterogeneous distributed database systems for production use. ACM Comput. Surv. 22, 237–266 (1990)
15. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. 37, 42–81 (2005)
16. Yu, H., Vahdat, A.: Design and evaluation of a continuous consistency model for replicated services. In: Proceedings of Operating Systems Design and Implementation, pp. 305–318 (2000)
17. Santos, N., Veiga, L., Ferreira, P.: Vector-field consistency for ad-hoc gaming. In: Cerqueira, R., Campbell, R.H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 80–100. Springer, Heidelberg (2007)
18. Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M.J., Theimer, M.M., Welch, B.B.: Session guarantees for weakly consistent replicated data. In: Proceedings Third International Conference on Parallel and Distributed Information Systems, Austin, Texas, pp. 140–149 (1994)
19. Baldoni, R., Guerraoui, R., Levy, R.R., Quéma, V., Tucci Piergiovanni, S.: Unconscious Eventual Consistency with Gossips. In: Datta, A.K., Gradinariu, M. (eds.) SSS 2006. LNCS, vol. 4280, pp. 65–81. Springer, Heidelberg (2006)
20. Ratner, D., Reiher, P., Popek, G.: Roam: A scalable replication system for mobile computing. In: DEXA 1999: Proceedings of the 10th International Workshop on Database & Expert Systems Applications, Washington, DC, USA, p. 96. IEEE Computer Society, Los Alamitos (1999)
21. Golding, R.: Modeling replica divergence in a weak-consistency protocol for global-scale distributed data bases. Technical Report UCSC-CRL-93-09, UC Santa Cruz (1993)
22. Fekete, A., Gupta, D., Luchangco, V., Lynch, N.A., Shvartsman, A.A.: Eventually-serializable data services. In: Symposium on Principles of Distributed Computing, pp. 300–309 (1996)
23. Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: Flexible update propagation for weakly consistent replication. In: Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16), Saint Malo, France (1997)
24. Pâris, J.F., Long, D.D.E.: Efficient dynamic voting algorithms. In: Proceedings of the Fourth International Conference on Data Engineering, pp. 268–275. IEEE Computer Society, Washington (1988)
25. Jajodia, S., Mutchler, D.: Dynamic voting algorithms for maintaining the consistency of a replicated database. ACM Transactions on Database Systems 15, 230–280 (1990)
26. Amir, Y., Wool, A.: Evaluating quorum systems over the internet. In: Symposium on Fault-Tolerant Computing, pp. 26–35 (1996)
27. Keleher, P.: Decentralized replicated-object protocols. In: Proc. of the 18th Annual ACM Symp. on Principles of Distributed Computing (PODC 1999) (1999)
28. Cetintemel, U., Keleher, P.J., Bhattacharjee, B., Franklin, M.J.: Deno: A decentralized, peer-to-peer object replication system for mobile and weakly-connected environments. IEEE Transactions on Computer Systems (TOCS) 52 (2003)
29. Çetintemel, U., Keleher, P.J., Franklin, M.J.: Support for speculative update propagation and mobility in deno. In: IEEE International Conference on Distributed Computing Systems (ICDCS), pp. 509–516 (2001)

30. Holliday, J., Steinke, R., Agrawal, D., Abbadi, A.E.: Epidemic algorithms for replicated databases. IEEE Transactions on Knowledge and Data Engineering 15, 1218–1238 (2003)
31. Barreto, J., Ferreira, P.: Version vector weighted voting protocol: efficient and fault-tolerant commitment for weakly connected replicas. Concurrency and Computation: Practice and Experience 19, 2271–2283 (2007)
32. Kistler, J.J., Satyanarayanan, M.: Disconnected operation in the Coda file system. In: Proceedings of 13th ACM Symposium on Operating Systems Principles, ACM SIGOPS, pp. 213–225 (1991)
33. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21, 558–565 (1978)
34. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications, vol. 2, pp. 483–502 (2002)
35. Barreto, J., Ferreira, P., Shapiro, M.: Exploiting our computational surroundings for better mobile collaboration. In: 8th International Conference on Mobile Data Management (MDM 2007), pp. 110–117. IEEE, Los Alamitos (2007)