Characterizing the Consistency of Online Services (Practical Experience Report)

Filipe Freitas[‡†⋆], João Leitão[†], Nuno Preguiça[†], and Rodrigo Rodrigues[⋆*]
[‡]*ISEL, Instituto Superior de Engenharia de Lisboa, Portugal;* [†]*NOVA-LINCS & FCT, Universidade NOVA de Lisboa, Portugal;* [⋆]*INESC-ID;* [*]*IST, Universidade de Lisboa, Portugal*

***Abstract*—While several proposals for the specification and implementation of various consistency models exist, little is known about what is the consistency currently offered by online services with millions of users. Such knowledge is important, not only because it allows for setting the right expectations and justifying the behavior observed by users, but also because it can be used for improving the process of developing applications that use APIs offered by such services. To fill this gap, this paper presents a measurement study of the consistency of the APIs exported by four widely used Internet services, the Facebook Feed, Facebook Groups, Blogger, and Google+. To conduct this study, our work (1) proposes definitions for a set of relevant consistency properties; (2) develops a simple, yet generic methodology comprising a small number of tests, which probe these services from a user perspective, and try to uncover consistency anomalies that are key to our definitions; and (3) reports on the analysis of the data obtained from running these tests for a period of several weeks. Our measurement study shows that some of these services do exhibit consistency anomalies, including some behaviors that may appear counter-intuitive for users, such as the lack of session guarantees for write monotonicity.**

I. Introduction

In recent years, we have seen an explosion in the use of services where multiple users, spread across the globe, interact and share content. Examples range from online social networks to content sharing and messaging services, and include services such as Twitter, Facebook, Google+, LinkedIn, or Instagram.

The infrastructure that forms the back-end for these services normally makes use of geo-replication for both dependability and good performance [1], [2]. The dependability motivation stems for the ability to tolerate catastrophic failures by having data replicated at multiple sites, whereas the performance gains come from being able to direct users to nearby copies of the data they want to access.

The price to pay for geo-replication is that the designers of these infrastructures have to deal with an inherent trade-off between performance and consistency [3]. If they choose to provide a strongly consistent access to the service, coordination among replicas at different sites becomes a necessity, increasing the latency for request execution. In contrast, if they choose to provide weak consistency, then operations can execute by contacting a single replica, but the semantics of the service will differ from those of a centralized server.

In this paper, we try to understand what is the consistency that is offered by the APIs of some of these Internet services. This is important for two main reasons. First, this allows us to better explain and understand the situations where the behavior of the service may be counter-intuitive. Second, this is important to help developers who design applications that interact and make use of these services, to know what they can expect when using these APIs. This allows those programmers to anticipate the effects of using the service on their applications, and to determine if they need to introduce additional logic to mask any undesirable behavior.

For understanding the consistency levels of online service APIs, this paper presents the results of a measurement study of the consistency of three popular platforms: Facebook, Google+, and Blogger. Our methodology for conducting this study started by identifying a set of anomalies that are not allowed by several consistency definitions. We then designed two black box tests that probe a given system (through its API) in search of manifestations of these anomalies. We implemented these tests and conducted an extensive consistency measurement experiment in the platforms mentioned above, including two variants of the Facebook API: Facebook Feed and Facebook Group services.

Our study lasted for about a month for each service with a total of 8183 individual tests being executed. Our main findings can be summarized as follows. We found a large prevalence of consistency anomalies in all services with the exception of Blogger. Furthermore, Google+ and Facebook Feed exhibited all anomalies we consider, whereas Facebook Groups exhibited only a subset of them. A large fraction of these anomalies can be masked with client-side techniques that do not require blocking user requests waiting for cross-replica synchronization.

The remainder of the paper is organized as follows. We survey related work in Section II. We define the anomalies used by our methodology in Section III, and the tests to determine their presence in Section IV. We present our measurement study in Section V, and conclude in Section VI.

II. Related Work

The most closely related paper is the very recent work of Lu et al. [4], which studied the consistency of TAO, Facebook's graph store. The study was performed by logging information inside the infrastructure of Facebook. In contrast, our approach uses the Web APIs of the services,

allowing to study the consistency of services, as perceived by end users, without access to their infrastructure. The other main distinction is that our methodology allowed us to study several Internet services, instead of a single one.

Previous authors have conducted studies on the consistency of data storage services. In particular, Wada et al. [5] have focused on testing the properties of *read-your-writes*, *monotonic reads*, and *monotonic writes* on several cloud storage services, namely Amazon SimpleDB, Amazon S3, Google App Engine, Microsoft Azure Table, and Blob Storage. Their study has focused on how the semantics differ depending on the relative location of readers and writers (same or different threads, processes, virtual machines, or geographical regions). Another relevant study in this context was conducted by Bermbach et al. [6], focusing on the consistency guarantees of Amazon S3 under a heavy load of concurrent writes. In contrast to both prior studies, our measurement study focuses on understanding the consistency properties offered by service APIs (i.e., above the storage layer), and for the particular case of clients scattered across different geographic locations. At an even lower layer, Xu et al. [7] conducted a measurement study of response times of virtual machines launched at Amazon EC2. This represents a layer that is even further apart from the one we are analyzing. Furthermore, that study focuses only on performance and not on consistency. Finally, our own short position paper motivates our work and presents a simple preliminary experiment [8].

Other papers have proposed analytic models to determine the consistency properties implemented by distributed key-value stores, based on measurements taken from inside the system. Anderson et al. [9] infer the consistency properties offered by key-value stores through the construction of graphs that capture the operations and their return values, to detect violations of the consistency levels defined by Lamport [10]. Zellag et al. [11] follow a similar approach, building a graph capturing the operations over a system. This graph is enriched with dependencies among object versions for detecting particular consistency anomalies. In contrast, we conduct a measurement study of consistency properties offered by Internet services, focusing on a more general mode than key-value stores.

Prior work defined a continuous degree of consistency. Bailis et al. [12] model the consistency of weak quorums as a probabilistic bound on staleness. Yu and Vahdat [13] limit the divergence to the final replica state. Part of our analysis builds on the idea of quantifying divergence but, in contrast, our goal is to understand its existence in several APIs.

III. Consistency Anomalies

The goal of this work is to characterize the consistency offered by online service APIs. A particular challenge in this context is that there are multiple consistency definitions, often using different notations. To address this, we define a number of anomalies that are both precise and intuitive to understand by programmers and users of online services. Note that we are not trying to exhaustively define all anomalies that can occur, nor to prove that these are equivalent to any of the various existing consistency definitions. It is also important to point out that if an anomaly is not observed in our tests, this does not imply that the implementation disallows for its occurrence, since it could have been by chance that it did not surface during our tests.

In the following description, we consider that users (or clients) of the service issue a set of requests, which can be divided into two categories: (1) *write* requests, which create an event that is inserted into the service state (e.g., posting a new message), and (2) *read* requests, which return a sequence of events that have been inserted into the state (e.g., reading the current sequence of posts). For simplicity, we assume that read operations return the entire sequence of writes. In practice, this could be generalized to define that a read returns a value that is a function of the sequence of writes according to some service specification, that may not contain the whole sequence.

Defining consistency anomalies: Based on these write and read operation categories, we now define the set of anomalies we consider in this study. We split these into three categories:

1) Session guarantees: The first set of anomalies corresponds to violations of session guarantees [14].

Read Your Writes: This session guarantee requires that a read observes all writes previously executed by the same client. More precisely, say W is the set of write operations made by a client c at a given instant, and S a sequence (of effects) of write operations returned in a subsequent read operation of c, a *Read Your Writes* anomaly happens when:

$$\exists x \in W : x \notin S$$

Monotonic Writes: This requires that writes issued by the same client are observed in the order in which they were issued. More precisely, if W is a sequence of write operations made by client c up to a given instant, and S is a sequence of write operations returned in a read operation by any client, a *Monotonic Writes* anomaly happens when the following property holds, where $W(x) \prec W(y)$ denotes x precedes y in sequence W:

$$\exists x, y \in W : W(x) \prec W(y) \wedge y \in S \wedge (x \notin S \vee S(y) \prec S(x))$$

Monotonic Reads: This session guarantee requires that all writes reflected in a read are also reflected in all subsequent reads performed by the same client. In comparison to monotonic writes, this has the subtle difference of requiring that the missing write is first observed (i.e., returned by a previous read) by the client before disappearing. More precisely, a *Monotonic Reads* anomaly happens when a client c issues two read operations that return sequences S_1 and S_2 (in that order) and the following property holds:

$$\exists x \in S_1 : x \notin S_2$$

Writes Follows Reads: This session guarantee requires that the effects of a write observed in a read by a given client always precede the writes that the same client subsequently performs. This precludes the situation where a client reacts to a write issued by itself or some other client (e.g., after reading a question that was posed) by issuing another write (e.g., posts a reply), and subsequently some client observes the second write without observing the first one. More precisely, if S_1 is a sequence returned by a read invoked by client c, w a write performed by c after observing S_1, and S_2 is a sequence returned by a read issued by any client in the system; a *Writes Follows Reads* anomaly happens when:

$$w \in S_2 \wedge \exists x \in S_1 : x \notin S_2$$

Note that although this last anomaly has been used to exemplify causality violations in previous papers [15], [16], any of the previous anomalies represent a different form of a causality violation [14].

2) Divergence: The next two anomalies refer to divergence between the state that is returned by read operations issued by two independent clients.

Content Divergence: A content divergence anomaly captures the case where two clients issue read operations and there are at least two writes such that one client sees one but not the other, and for the other client the opposite is true. More precisely, a content divergence anomaly happens when two reads issued by clients, c_1 and c_2, return respectively, sequences S_1 and S_2, and the following property holds:

$$\exists x \in S_1, y \in S_2 : x \notin S_2 \wedge y \notin S_1$$

Any system relying on weak consistency protocols is prone to this anomaly, as this is a consequence of the core property of being able to perform and complete a write operation by contacting a single replica in the system.

Order Divergence: The order divergence anomaly refers to writes issued by different clients being seen in reverse orders by different clients. More precisely, an order divergence anomaly happens when two reads issued by two clients, c_1 and c_2, return sequences S_1 and S_2, containing a pair of events occurring in a different order at the two sequences:

$$\exists x, y \in S_1, S_2 : S_1(x) \prec S_1(y) \wedge S_2(y) \prec S_2(x)$$

where $S(x) \prec S(y)$ means that operation x in state S was ordered before operation y.

3) Quantitative metrics: The anomalies defined so far are boolean predicates over a trace of the system, i.e., they either occur in an execution or they do not. In addition to the presence or absence of these anomalies, we can determine quantitative aspects of the observed behavior.

Content Divergence Window and Order Divergence Window: When considering the two divergence anomalies, it is also relevant to understand how long it takes for the system to converge back to a single coherent state. As such, we can define the *Content Divergence Window* and *Order Divergence Window* as follows. When a set of clients issue a set of write operations, the divergence window is the amount of time during which the condition that defines the anomaly (either content or order divergence) remains valid, as perceived by the various clients.

IV. Measurement methodology

In this section, we describe a methodology for testing online services that tries to expose the previously defined anomalies. At a high level, our methodology consists of deploying agents in different points in the globe, who perform several black box tests of the online service, by issuing several reads and writes to the service.

The notion of a read or a write operation is specific to each service, but should adhere to the specification in Section III. For the services we analyze, since they are either social networking or messaging services, we chose operations that posted a message and listed the current sequence of posts.

Time synchronization: An important aspect of our tests is that they require the clocks of the machines where agents are deployed to be loosely synchronized, for two different reasons. First, we use clock readings to compute divergence windows between different agents. As such, a clock synchronization error could introduce an additional error in the computed values for the divergence windows. Second, some of the tests require the various agents to issue operations as simultaneously as possible (namely to maximize the chances of triggering divergence). As such, a synchronization error would decrease the chances of triggering divergence.

To synchronize clocks, one could rely on a service such as NTP. However, the use of NTP implies releasing the control over the clock synchronization process, which could introduce errors in our measurements when the clock is adjusted. Thus, we disabled NTP and implemented a simple protocol that estimates the time difference between a local clock and a reference clock (resembling Cristian's algorithm for clock synchronization [17].). In particular, a coordinator process conducts a series of queries to the different agents to request a reading of their current local time, and also measures the RTT to fulfill that query. The clock deltas are then calculated by assuming the time spent to send the request and receive the reply are the same, and taking the average over all the estimates of this delta. The uncertainty of this computation is half of the RTT values.

Tests: Our goal in designing these tests is twofold: first, we want the tests to be complete, in the sense that they allow (and even promote) the possibility of exposing all listed anomalies; and second, we want to make these simple and limit the number of different tests required. Guided by these principles, we designed the following two tests.

4) Test 1: The sequence of events for the first test is depicted in Figure 1. In this test, each agent performs two consecutive writes and continuously issues reads in the background, with a frequency that is determined by the maximum frequency that is allowed by the rate limit of the online service. The writes by the different agents are staggered: agents have sequential ids and the first write by

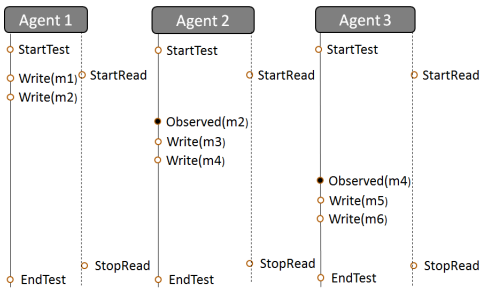


Figure 1. Timeline for Test 1 with three agents.

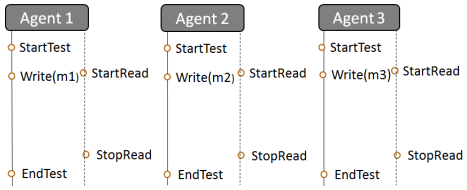


Figure 2. Timeline for Test 2 with three agents.

agent i is issued when it observes the last write of agent $i-1$. For all operations, we log the time when they occurred (invocation and response times) and their output.

The output of running this test already allows us to detect most of the consistency anomalies from the previous section:

• A violation of Read Your Writes occurs, for instance, when Agent 1 writes M1 (or M2), and in a subsequent read operation M1 (or M2) is missing. (The same applies to each message written by the remaining agents.)

• A violation of Monotonic Writes occurs, for instance, when Agent 1 writes M1 and M2, and afterwards that agent either observes only the effects of M2 in the output of a read operation, or observes the effect of both writes in a different order. (The same applies to each pair of messages written by the remaining agents.)

• A violation of Monotonic Reads occurs when any agent observes the effect of a message M and in a subsequent read by the same agent the effects of M are no longer observed.

• A violation of Writes Follows Reads occurs when any agent either observes M3 without observing M2 or observes M5 without observing M4. We only consider these particular pairs of messages because, in the design of our test, M3 and M5 are the only write operations that require the observation of M2 and M4, respectively, as a trigger.

5) Test 2: The timeline for the second test is depicted in Figure 2. This test attempts to uncover divergence among the view that different agents have of the system, by having all agents issue a single write (roughly) simultaneously, and all agents continuously reading the current state in the background. This simultaneity could increase the chances of different writes arriving at different replicas in a different order, and therefore trigger divergence.

The output of running this test gauges the remaining questions from the previous section. In particular:

• A violation of Content Divergence occurs, for instance, when an Agent observes a sequence of writes containing only M1 and another Agent sees only M2.

• A violation of Order Divergence occurs, for instance, when an Agent sees the sequence (M2,M1) and another Agent sees the sequence (M1,M2).

The respective windows are also computed using the results of this test by ordering all events according to their absolute time (factoring in the correction for clock deltas as explained previously), and determining the interval of time during which the anomaly conditions hold, as determined by the most recent read. Note that the timeline considering operations from all agents may lead to the following situation: agent 1 reads (M1) at time t_1; agent 1 reads (M1,M2) at t_2; agent 2 reads (M2) at t_3; agent 2 reads (M1,M2) at t_4, with $t_1 < t_2 < t_3 < t_4$. Although there has been a content divergence anomaly, the computed window is zero.

V. Measurement Study

In this section, we present the results of our measurement study of four online service APIs: Google+, Blogger and two services from Facebook.

For Blogger, we used the API to post blog messages and to obtain the most recent posts. In this service, each agent was a different user, and all agents wrote to a single blog.

For Facebook we used the Facebook Graph API to interact with both the user news feed and group feed services. In the first case, each user wrote to and reads from his own feed, which combines writes to the user feed and from the feeds of all friends. In the second case, all users are associated with a single group and issued all their write and read operations over that group. In all tests, each agent was a different user, and we used test users, which are accounts that are invisible to real user accounts. We conducted a small number of tests with regular accounts and results were consistent with those of test users.

For Google+, we could only find API support for posting "moments". We used the API to post a new moment and to read the most recent moments. In this case, all agents shared the same account, since there is no notion of a follower for moments. Unfortunately, there does not seem to exist an extensive use of the concept of moments, particularly since these are not easy to use through the Web interface. As such, we cannot know whether the results we present apply to what most users observe when they access Google+ services. Note that the behavior of a service accessed through the API might differ when using the Web Interface.

Subsequently to the experiments, Google+ disabled its support for moments, and Facebook removed the ability to read user news feed from their Graph API.

For each of these services, we ran three agents, which were deployed on three geographically distant locations using different "availability zones" of Amazon EC2. In particular, we used the Oregon (US), Tokyo (Japan), and Ireland availability zones. Furthermore, we deployed a coordinator

	Google+	Blogger	FB Feed	FB Group
Period between reads	300ms	300ms	300ms	300ms
Number of reads per agent per test (average)	48	11	14	11
Time between successive tests	34min	20min	5min	5min
Number of tests executed	1036	1028	1020	1027

Table I
CONFIGURATION PARAMETERS FOR TEST 1

	Google+	Blogger	FB Feed	FB Group
Period between reads	300ms (14X) then 1s	300ms (13X) then 1s	300ms (20X) then 1s	300ms (20X) then 1s
Reads per agent per test	$17-75$	20	40	50
Time between successive tests	17min	10min	5min	5min
Number of executed Tests	922	1012	1012	1126

Table II
CONFIGURATION PARAMETERS FOR TEST 2

in another availability zone, in this case North Virginia (US). The average RTT values measured between the coordinator and the remaining agents are: 136ms to Oregon; 218ms to Japan; and 172ms to Ireland. These were employed to derive the clock deltas, as discussed in the previous section.

For each of the services, we deployed the various agents for a total period of roughly 30 days per service (for running both tests). For each service, we alternated between running each of the two test types roughly every four days: instances of test 1 ran repeatedly for four days then test 2 for another four days then back to test 1, and so on. Each instance of a test ran until completion: for test 1, the test is complete when all agents see the last message written by Agent3 (M6), and for test 2, the test completes when all agents perform a configurable number of reads. Due to rate limits, after a test instance finishes, we had to wait for a fixed period of time before starting a new one. Before the start of each iteration of a test, the clock deltas were computed again.

Tables I and II summarize the parameters we used for configuring each of the tests for each service. These parameters were chosen in a way that minimizes the time to collect the data while taking into consideration rate limits for each service. For the second test, the period between reads is adaptive: initially it is short, and then it becomes one second. This allows for a higher resolution in the period when the writes are more likely to become visible, while respecting the rate limits. In total, we ran $1,958$ tests comprising $323,943$ reads and $8,982$ writes on Google+, $2,040$ tests comprising $96,979$ reads and $9,204$ writes on Blogger, $2,032$ tests comprising $195,029$ reads and $9,156$ writes on Facebook Feed and $2,153$ tests comprising $169,299$ reads and $9,540$ writes on Facebook Group.

Overall results: We start by analyzing the overall prevalence of anomalies in both tests. Figure 3 shows, for each anomaly and each service, the percentage of tests with anomalies. All types of anomalies were seen in both Google+ and Facebook Feed, whereas in Facebook Group no violations of Read Your Writes and Order Divergence were seen. For the remaining anomalies, the most common

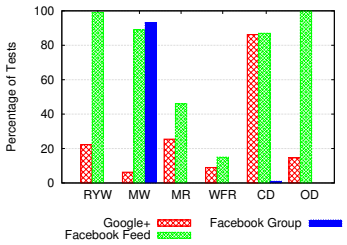
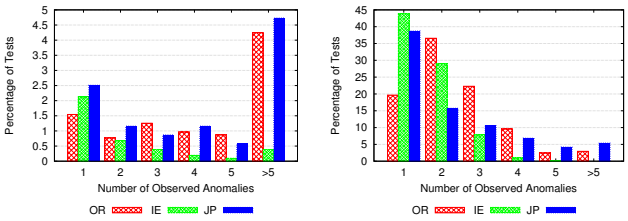
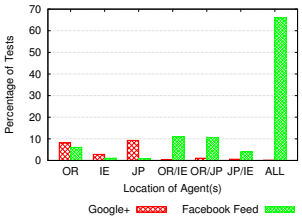


Figure 3. Percentage of tests with observations of different anomalies in Google+, Facebook Feed and Facebook Group

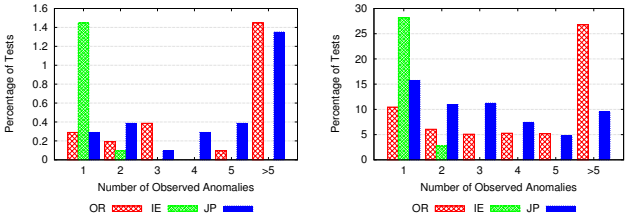


(a) Google+ (b) Facebook Feed

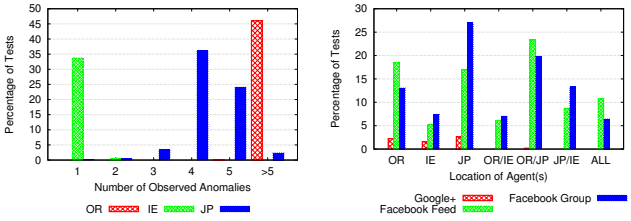


(c) Correlation of anomalies

Figure 4. Distribution of read your writes anomalies per test.



(a) Google+ (b) Facebook Feed



(c) Facebook Group (d) Correlation of anomalies

Figure 5. Distribution of monotonic writes anomalies per test.

in this service was Monotonic Writes. In Blogger we did not detect any anomalies of any type. Next, we analyze the occurrence of each anomaly in detail. We omit the results from the combinations of services and anomaly types where no anomalies were seen.

Session guarantees: We analyze the prevalence of anomalies for each session guarantee. For the **read your writes** guarantee, Figure 3 shows a high value (99%) for

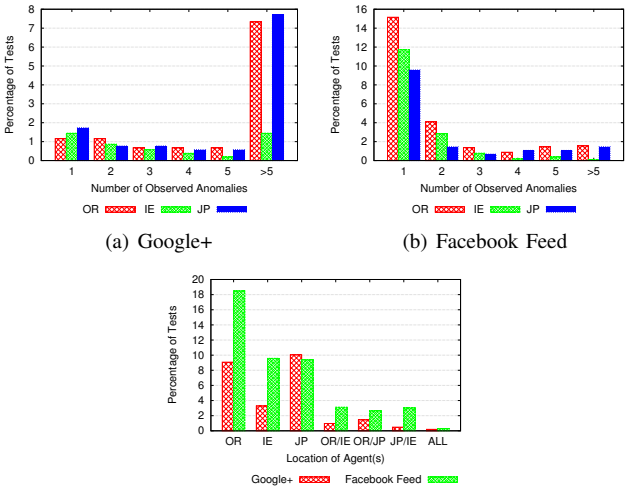
Facebook Feed and a visible presence of this type of anomaly (22%) in Google+. Figure 4(a) presents the number of observations of the anomaly per test for Google+. This shows that, in the particular case of Google+, more than half of the tests where this anomaly was detected had several individual violations of the property. The results also show that this anomaly is more prevalent on clients in Oregon and Japan. The results for Facebook Feed, which are reported in Figure 4(b), show the opposite trend: most occurrences of this anomaly are in tests where it is only detected once or twice per agent. In contrast with Google+, Facebook Feed showed a similar prevalence across client locations.

To determine whether these anomalies are correlated across locations, Figure 4(c) depicts the percentage of tests where these anomalies occurred in each agent exclusively versus across different combinations of the agents. The results show that this does not tend to be a global phenomenon: in Google+, the large majority of occurrences are only perceived by a single agent. However, for Facebook Feed, all three locations perceived the anomaly in a large fraction of tests, because this anomaly arises much more frequently.

Next, we analyze the prevalence of violations of the **monotonic writes** session guarantee, with Figure 3 showing a significant prevalence of this type of anomaly both in Facebook Feed and in Facebook Group, with a 89% and 93% prevalence, respectively. Google+ shows a fairly low prevalence with only 6%. The results in Figure 5, for Google+, show that this anomaly, when detected in a test, is often observed several times in that test. Additionally, Oregon and Japan have an increased incidence of this anomaly occurring multiple times in a single test, whereas in Ireland, when this anomaly is detected, it often occurs a single time in each test. This phenomenon however might be a consequence of the way that our tests are designed, as in test 1 Ireland is the last client to issue its sequence of two write operations, terminating the test as soon as these become visible. Thus, it has a smaller opportunity window for detecting this anomaly. This observation is supported by the fact that the same trend is observed in the results for the Facebook services, and by additional experiments that we have performed, where we rotated the location of each agent.

Figure 5(d) presents the correlation of the location of agents across the tests that observed the anomaly. The figure shows that this tends to be a local occurrence in Google+, where the anomaly is visible in only one of the locations, whereas in Facebook Feed and Group this anomaly tends to be global with a larger prevalence in Japan.

The large occurrence of these anomalies in the Facebook services motivated us to inspect more carefully these phenomena across these services. We noticed that in Facebook Feed, messages are often reordered across different read operations executed by each agent. However, for the particular case of Facebook Group, the reordering of messages occurred mostly in messages issued by the same agent,



(a) Google+ (b) Facebook Feed

(c) Correlation of anomalies

Figure 6. Distribution of monotonic reads anomalies per test.

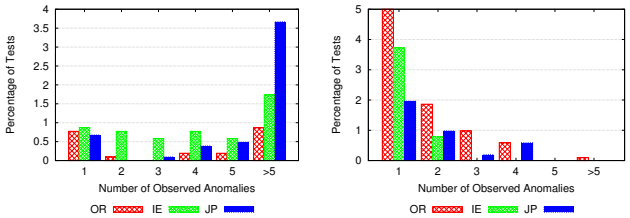
and that all agents observed this reordering of operations consistently. Upon further inspection, we noticed that each event in Facebook Group is tagged with a timestamp that has a precision of one second, and that whenever two write operations were issued by an agent within that interval (causing them to be tagged with the same timestamp) the effects of those operations would always be observed in reverse order. This suggests that, in this service, this anomaly is produced by a deterministic ordering scheme for breaking ties in the creation timestamp.

The experiment for **monotonic reads**, as shown in Figure 3, indicates that 46% of the tests are exhibiting this type of occurrence on Facebook Feed and 25% in Google+. This anomaly was detected in Facebook Group in a single test. Figure 6(a) shows a long tail in the number of observations per test in Google+. Although the anomaly is much more prevalent in Facebook Feed, the results show that it is mostly detected a single time per agent per test. Figure 6(c) indicates a mostly local phenomenon in both services.

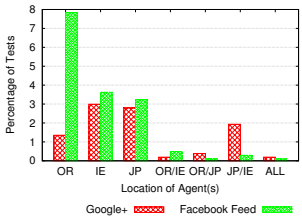
The last session guarantee, **writes follow reads**, is more frequent in Facebook Feed. As depicted in Figure 3, this anomaly only occurs in Facebook Group twice. Figure 7 shows that, although this is a somewhat frequent anomaly, it does not occur recurrently, with only a few observations occurring per agent in each test for Facebook Feed. In contrast, for Google+ we verify the opposite. Figure 7(c) depicts the set of agents that observe the lack of causality in each test. This indicates a mostly local phenomenon in both services, particularly located in Oregon for Facebook Feed.

Divergence: We now check the presence of divergence events in the collected data. We start by looking at **content divergence**. Figure 8 shows the percentage of tests with divergence between the state observed by pairs of agents.

These results show that content divergence occurs very frequently in Google+ and in Facebook Feed, which indicates the likely use of weakly consistent replication that privileges performance over strong consistency. In particular,

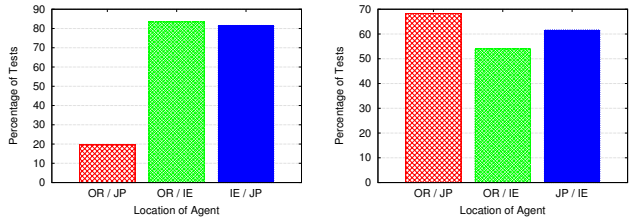


(a) Google+ (b) Facebook Feed

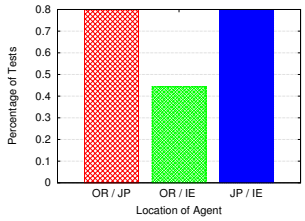


(c) Correlation of anomalies

Figure 7. Distribution of writes follow reads anomalies per test.



(a) Google+ (b) Facebook Feed



(c) Facebook Group

Figure 8. Percentage of tests with content divergence anomalies.

the percentage of tests that show any content divergence in Google+ is up to 85%, being less pronounced between Oregon and Japan than between the remaining pairs of agents. This might suggest that the Oregon and the Japan agents are connecting to the same data center, whereas the other pairs of agents are not. In the case of Facebook Feed, this occurrence is more uniform across all pairs of agents, and the prevalence is also high (above 50% across all pairs). In the case of Facebook Group the prevalence is extremely low with a total of 15 occurrences of this anomaly, 9 of which happened across a sequence of tests, where the Tokyo agent was unable to observe the operations of other agents. This suggests the agent in Tokyo connects to a different data center than the other agents, hence these anomalies might be caused by a transient fault or network partition.

In terms of the presence of **order divergence**, Figure 3 shows that this phenomenon occurs in Google+ and in Facebook Feed, with a prevalence that is less pronounced than content divergence in Google+. Similarly to the previously reported results, we observed that this anomaly is much less frequent between Oregon and Japan (below 1%) than between the remaining pairs of agents (around 14%). In Facebook Feed, the prevalence is near 100% at all locations.

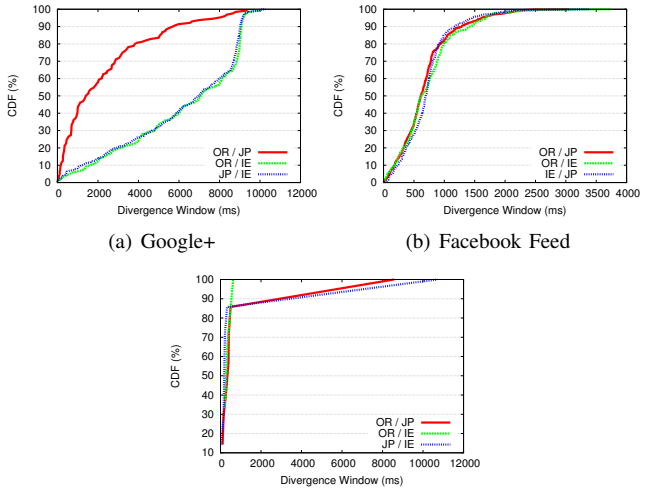
While the results for Facebook Feed may seem surprising, they are explained by the semantics of the service. This is because the reply to a read contains a subset of the writes, which are not the most recent ones, but a selection of writes based on a criteria that depends on the expected interest of these writes for the user issuing the read operation.

Quantitative metrics: Next, we analyze several quantitative measurements. Figure 9 shows the CDF of the **content divergence window**. This distribution is shown for each pair of agents and each service (only considering the largest divergence window for each pair of agents in each test). The results show a smooth distribution of this window, with Google+ taking substantially longer than the remaining services for all agents to regain a consistent view of the system state (on the order of seconds in Google+ versus hundreds of milliseconds in Facebook Feed and most of the content divergence instances observed in Facebook Group). Figure 9(a) shows that the Oregon and Japan agents have a convergence time that is much faster than the remaining pairs of agents, and that the other two pairs exhibit similar convergence times. This suggests that writes issued from Oregon and Japan may be processed by the same replica.

The results shown in Figure 9(b) show that content divergence occurs in Facebook Feed across all agents and all pairs of agents, and takes approximately the same time to converge to a consistent view of the service state (on the order of a few seconds). Again, the semantics of reads in this service may explain these results.

Finally, Figure 9(c) depicts the results for Facebook Group, showing that content divergence between the agent in Japan and the two remaining agents takes longer to be resolved. This indicates that the agent in Japan may be contacting a different replica than the remaining agents.

The last set of measurements refer to the **order divergence window**. This anomaly was only observed in Google+ and Facebook Feed. For Google+, Figure 10(a) shows that a coherent order is more quickly re-established between the Oregon and Japan agents than the remaining pairs, which can take over ten seconds to achieve this. The reason behind the steps in the curve is that, after the first 12–14 reads, which are more frequent, our agents perform reads with a period of one second, due to rate limiting. As such, the detection of the end of a window is done at the resolution of one second, and in a synchronized way, as shown in the CDF. For the Facebook Feed service we observe that a coherent order is established among the several pairs of agents faster. These results exclude runs where convergence was not reached during the test. The fraction of tests where this occurred was 81% for divergence between Oregon and Tokyo, 94% for Oregon and Ireland, and 89% for Tokyo and Ireland.



(a) Google+ (b) Facebook Feed

(c) Facebook Group

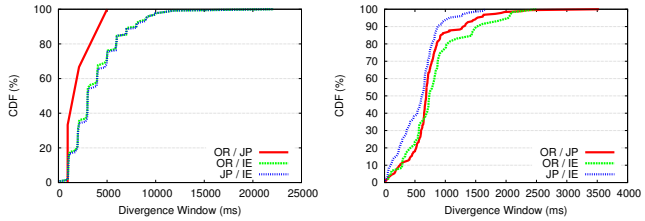
Figure 9. Cumulative distribution of content divergence windows.

Discussion of Results: Some of these results can be seen as a somewhat natural and even expected consequence of choosing performance over stronger consistency models such as serializability. In particular, when the designers of replication protocols choose to provide a fast response to the clients of the service after contacting only one or a small subset of replicas, the implication of this choice is that the consistency model offered by the service has to provide some form of weak consistency, namely one where content divergence is possible, since two writes that were processed by two different replicas may have executed without being aware of each other. We note that our results show an exception to this design principle in the Blogger system, which appears to be offering a form of strong consistency. This can be seen as a sensible design choice considering the write rate and user base size in Blogger. Overall, one of the most surprising results of our study was to find several types of anomalies that are not inevitable, even when choosing performance over consistency. This is the case of session guarantees, which previous work has shown how to implement, even under weak consistency [14].

Given that some of these services may not provide session guarantees, the relevant question is what provisions can applications make in order to handle this fact, especially if they are interested in providing these guarantees. It turns out that most of the session guarantees can be easily enforced at the application level by simply identifying requests with a session id and a sequence number within a session, and using a combination of caching and replaying previous values that were read and written, and delaying or omitting the delivery of messages. In contrast, the writes follows reads session guarantee is a bit more complicated to enforce.The details of such schemes are left as future work.

VI. Conclusions and Future Work

In this paper, we presented a measurement study of the consistency offered by the APIs of four online services. Our



(a) Google+ (b) Facebook Feed

Figure 10. Cumulative distribution of order divergence windows.

study shows the relatively frequent occurrence of most of the anomalies across all services except Blogger. This highlights the need for application writers to consider whether the intended semantics for their applications is compatible with these behaviors, and if not, to possibly write programs in a way that masks these anomalies. In the future, we would like to extend this methodology to other services, also considering white-box testing, so it can be applied to large-scale storage systems.

Acknowledgements: The research of Rodrigo Rodrigues is supported by the European Research Council under an ERC Starting Grant. This research was also supported in part by EU FP7 SyncFree project (609551), FCT/MCT SFRH/BD/87540/2012, PEst-OE/EEI/ UI0527/ 2014, NOVA LINCS (UID/CEC/04516/2013), and INESC-ID (UID/CEC/50021/2013).

References

[1] G. DeCandia et. al., "Dynamo: Amazon's Highly Available Key-value Store," in *Proc. of SOSP'07*.
[2] Brian F. Cooper et. al., "PNUTS: Yahoo!'s Hosted Data Serving Platform," *Proc. of VLDB'08*, vol. 1, no. 2.
[3] Kanat Tangwongsan et. al., "Parallel streaming frequency-based aggregates," in *Proc. of SPAA '14*.
[4] Haonan Lu et. al., "Existential consistency: Measuring and understanding consistency at facebook," in *Proc. of SOSP'15*.
[5] Hiroshi Wada et. al., "Data Consistency Properties and the Tradeoffs in Commercial Cloud Storages: the Consumer's Perspective," in *Proc. of CDIR'11*.
[6] D. Bermbach and S. Tai, "Eventual Consistency: How Soon is Eventual? An Evaluation of Amazon S3's Consistency Behavior," in *Proc. of MW4SOC'11*.
[7] Yunjing Xu et. al., "Bobtail: Avoiding Long Tails in the Cloud," in *Proc. of NSDI'13*.
[8] João Costa et. al., "Avaliação das Garantias de Constência em Serviços Geo-Replicados," in *Proc. of Inforum'13*.
[9] Eric Anderson et. al., "What consistency does your key-value store actually provide?" in *Proc. of HotDep'10*.
[10] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, 1998.
[11] K. Zellag and B. Kemme, "How consistent is your cloud application?" in *Proc. of the SOCC'12*.
[12] Peter Bailis et. al., "Probabilistically bounded staleness for practical partial quorums," *Proc. of VLDB'12*, vol. 5, no. 8.
[13] H. Yu and A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services," *ACM Trans. Comput. Syst.*, vol. 20, no. 3, 2002.
[14] Douglas B. Terry et. al., "Session guarantees for weakly consistent replicated data," in *Proc. of PDIS'94*.
[15] Wyatt Lloyd et. al., "Don't settle for eventual: Scalable causal consistency for wide-area storage with cops," in *Proc. of SOSP'11*.
[16] Sérgio Almeida et. al., "Chainreaction: A causal+ consistent datastore based on chain replication," in *Proc. of EuroSys'13*.
[17] F. Cristian, "Probabilistic clock synchronization," *Dist. Comp.*, vol. 3, no. 3, pp. 146–158, 1989.