On the degradation of distributed graph databases with eventual consistency

Paul Ezhilchelvan[1], Isi Mitrani[1], and Jim Webber[2]

[1] School of Computing, Newcastle University, NE4 5TG, UK
e-mail: paul.ezhilchelvan@ncl.ac.uk, isi.mitrani@ncl.ac.uk
[2] Neo4j UK, 41-45 Blackfriars Rd, London SE1 8NZ
e-mail: Jim.Webber@neo4j.com

Abstract. The 'eventual consistency' approach to updates in a distributed graph database leaves open the possibility that edge information may be corrupted. The process by which this occurs is modeled, with the aim of estimating the time that it takes for the database to become corrupted. A fluid approximation is developed and is evaluated for different parameter settings. Comparisons with simulations show that the results are very accurate.

1 Introduction

Managing distributed data typically means adopting either strongly or eventually consistent update policies. In the former approach, replicas of a data item are updated in an identical order at all servers [7]. When update requests are launched concurrently, users will see an identical update sequence, irrespective of the actual physical server they use for accessing the data. However, this single-server abstraction imposes a significant performance penalty, since update requests need to be ordered (and replicated) prior to being acted upon. Further, according to the CAP theorem (see [2, 6]), if the network partitions the servers, request ordering and access to data may be interrupted and the availability of services can be reduced.

In view of these disadvantages, large distributed data stores such as Google Docs, Dynamo [5] and Cassandra [4], opt instead for an eventually consistent update policy (see [10]). Update requests are processed as soon as they arrive. This enhances system performance and availability but leaves a time window in which values of a replicated data item at different servers can be mutually inconsistent. These windows may occasionally lead to incorrect data operations. Eventual consistency is a viable model for applications where availability is paramount and where the consequences of any incorrect operations can be dealt with through compensations and state reconciliations.

For example, Bailis and Ghodsi [1] refer to an ATM service where eventual consistency can allow two users to simultaneously withdraw more money than their (joint) bank account holds; such an anomaly, on being detected, is reconciled by invoking exception handlers. Given that an ATM service is expected to be available 24/7, the eventually consistent approach is appropriate.

In this paper, we focus on the effects of adopting the eventually consistent policy in systems where occasional incorrect operations are not immediately or readily observed, and can lead to large scale propagation of erroneous states. Reconciliation at a later time can become impossible. The systems of interest here are Graph Databases (see Robinson, Webber and Eifrem, [9]), which are a rapidly growing database technology at present.

A graph database consists of *nodes* and *edges*, representing entities and relations between them, respectively. For example, node A may represent a person of type Author and B an item of type Book. A and B will have an edge between them if they have a relation, e.g. A is an author of B. The popularity of the graph database technology owes much to this simple structure from which sophisticated models can be easily built and efficiently queried. Examples of operations performed on a graph database are: counting the number of fans following a famous person on Twitter, ranking the most frequently accessed pages in the web, etc.

When nodes are connected by an edge, the database stores some reciprocal information. For example, if there is an edge between A and B, then A would have a field *wrote B* and B would have a field *written by A*; similarly, if there is an edge between a music fan F and a singer S, then they would have the reciprocal fields *following S* and *being followed by F*. Storing this reciprocal information is a critical design decision. In a distributed graph database this is a non-trivial problem since the information at nodes across different servers must remain mutually compatible. Any updates in one connected node at one server must be reflected in the other node(s) at a different server(s).

When a query writes a distributed edge, the eventual consistency policy cannot ensure that the necessary updates are implemented in a timely or consistently-ordered fashion across the servers involved. This can cause major problems. Suppose that two queries operate (nearly) simultaneously on a given distributed edge, each starting from a different server. Then the two updates can be implemented in a different order at the two servers, leading to a mutual incompatibility between two nodes. While such a state exists, there may be a stream of queries reading one node and another stream reading the other. One of the two streams is obviously reading incorrect information (from a global point of view).

A query that reads incorrect information about one edge and updates another edge, introduces incorrect information at both nodes of the second edge. Those two nodes may then be mutually compatible, yet incorrect. Such errors cannot be detected by simple compatibility tests. Moreover, they are propagated throughout the database by subsequent queries that carry out updates based on incorrect information. Eventually, the quality of information held by the database becomes so degraded that the database becomes unusable.

The contribution of this paper is to model the above process of degradation. This has not, to the best of our knowledge, been done before. This work is thus the first in formally assessing the damage that the eventually consistent update policy can inflict on a distributed graph database.

We provide easily implementable, efficient and accurate solutions that allow us to determine the time it takes for the database to lose its utility. These

solutions are then used in experiments aimed at examining quantitatively the effect that various parameters have on the degradation process. At the same time, the accuracy of the estimates is evaluated by comparisons with simulations.

The model is described in section 2. Section 3 develops the solutions, based on fluid approximations. The numerical and simulation results are presented in section 4.

2 The model

A popular implementation of a graph database contains, for each node, a list of adjacency relations describing the incoming and outgoing edges associated with that node. When an edge is updated, the corresponding entries in both the origin and the destination nodes must be updated. If those two nodes are stored on the same server, then the edge is said to be 'local'. A local update is assumed to be instantaneous. An edge connecting two nodes stored on different servers is said to be 'distributed'. A 'write' operation for a distributed edge is carried out first on one of its servers and then, after a small but non-zero delay, on the other.

This implementation of distributed writes makes it possible to introduce faults in edge records. Consider an edge e, spanning two servers, S_1 and S_2. A query Q_1, containing a write operation for e, arrives in S_1 at time t and is performed in S_2 at time $t+\delta$. At some point between t and $t+\delta$, another query, Q_2, also writing e, arrives in S_2 and is performed in S_1 some time later. The result of this occurrence, which will be referred to as a 'conflict', is that the S_1 entry for e is written in the order Q_1, Q_2, while the S_2 entry is written in the order Q_2, Q_1. One of these entries may be considered correct, but an external observer cannot tell which is which. Such edges are called 'half-corrupted'.

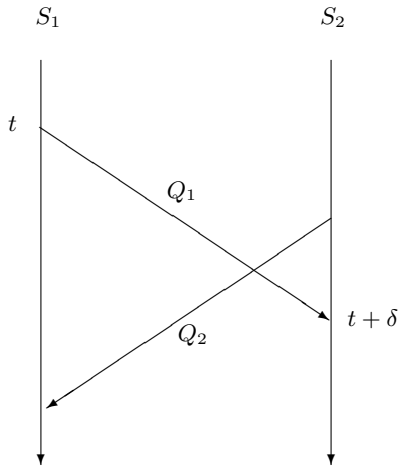

Fig. 1. Conflict between Q_1 and Q_2

The mechanism of conflict is illustrated in Figure 1, where the time at S_1 and S_2 is shown flowing downwards.

A subsequent query which happens to read the correct entry of a half-corrupted edge, and completes a write operation for it without a conflict, will repair the fault and make the edge record clean again. However, if it reads the incorrect entry and writes any edge, it causes the target to become 'semantically corrupted', or simply 'corrupted'. The correct and incorrect reads are equally likely, so each occurs with probability 1/2.

Any edge can become corrupted by being written on the basis of reading incorrect information. Corrupted edges cannot be repaired, since there is no post-facto solution to the graph repair problem in the general case.

Queries that update edges arrive in a Poisson stream with rate λ per second. We assume that each query contains a random number of read operations, K, followed by one write operation. This is a conservative assumption, since more than one writes per query would increase the rate of corruption. The variable K can have an arbitrary distribution, $P(K = k) = r_k$. In practice K tends to be at least 2, i.e. $r_0 = r_1 = 0$. The K edges read, and the one written by the query are assumed to be independent of each other (but note below that they are not equally likely).

The edges in the database are divided into T types, numbered 1, 2, ..., T in reverse order of popularity. The probability that a read or a write operation accesses an edge of type i is p_i, with $p_1 > p_2 > \ldots > p_T$. The number of edges of type i is N_i, and typically $N_1 < N_2 < \ldots < N_T$. The total number of edges is N. In every type, a fraction f of the edges are distributed and the rest are local. The probability of accessing a particular edge of type i, for either reading or writing, is p_i/N_i.

At time 0, all edges are clean (free from corruption). When a certain fraction, γ (e.g., $\gamma = 0.1$), of all edges become corrupted, the database itself is said to be corrupted for practical purposes. The object of the analysis is to provide an accurate estimate of the length of time that it takes for this to happen.

At any moment in time, an edge belongs to one of the following four categories.

Category 0: Local and clean.
Category 1: Distributed and clean.
Category 2: Half-corrupted.
Category 3: Corrupted.

Only distributed edges can be in category 2, but any edge, including local ones, can be in category 3.

Denote by $n_{i,j}(t)$ the number of type i edges that are in category j at time t. The set of vectors $\mathbf{n}_i(t) = [n_{i,0}(t),n_{i,1}(t),n_{i,2}(t),n_{i,3}(t)]$, for $i = 1, 2, \ldots, T$, define the state of the database at time t. At all times, the elements of vector $\mathbf{n}_i$ add up to N_i. Any state such that

$$\sum_{i=1}^{T} n_{i,3}(t) \geq \gamma N \,, \tag{1}$$

will be referred to as an 'absorbing state'. The absorbing states correspond to a corrupted database.

The value of interest is U, the average first passage time from the initial state where $\mathbf{n}_i(0) = [(1-f)N_i, fN_i, 0, 0]$ (i.e., a clean database), to an absorbing state.

The above assumptions and definitions imply that a read operation performed at time t would return a correct answer with probability α, given by

$$\alpha = \sum_{i=1}^{T} \frac{p_i}{N_i}[n_{i,0}(t) + n_{i,1}(t) + \frac{1}{2}n_{i,2}(t)] \ . \tag{2}$$

The probability, β, that all the read operations in a query arriving at time t return correct answers, is equal to

$$\beta = \sum_{k=1}^{\infty} r_k \alpha^k \ . \tag{3}$$

Suppose that the distribution of K is geometric with parameter r, starting at $k = 2$; in other words, $r_k = (1-r)^{k-2}r$. Then the above expression becomes

$$\beta = \frac{\alpha^2 r}{1 - \alpha(1-r)} \ . \tag{4}$$

Consider now the probability, q_i, that a query of type i arriving at time t and taking a time δ to complete a write operation, will be involved in a conflict. That is the probability that another query of type i arrives between t and $t + \delta$ and writes the same edge, but starting at its other end. This can be expressed as

$$q_i = 1 - e^{-\frac{1}{2}\lambda p_i \delta / N_i} \ \ ; \ \ i = 1, 2, \ldots, T \ . \tag{5}$$

Note that in practice δ is dominated by network delays. The actual processing time associated with a write operation is negligible.

If the time to complete a distributed write is not constant, but is distributed exponentially with mean δ, then the conflict probability would be

$$q_i = \frac{\lambda p_i \delta}{2N_i + \lambda p_i \delta} \ \ ; \ \ i = 1, 2, \ldots, T \ . \tag{6}$$

When δ is small, there is very little difference between these two expressions.

An incoming query that is involved in a conflict would change the category of a distributed edge from 1 to 2, provided that all read operations of both queries return correct results. Hence, the instantaneous transition rate, $a_{i,1,2}$, from state $[n_{i,0}, n_{i,1}, n_{i,2}, n_{i,3}]$ to state $[n_{i,0}, n_{i,1} - 1, n_{i,2} + 1, n_{i,3}]$, can be written as

$$a_{i,1,2} = \frac{\lambda p_i n_{i,1}}{N_i} q_i \beta^2 \ . \tag{7}$$

Conversely, an incoming query writing a category 2 edge can change it to a category 1 edge, provided that all its read operations return correct results and

it is not involved in a conflict. Hence, the instantaneous transition rate, $a_{i,2,1}$, from state $[n_{i,0},n_{i,1},n_{i,2},n_{i,3}]$ to state $[n_{i,0},n_{i,1}+1,n_{i,2}-1,n_{i,3}]$, is given by

$$a_{i,2,1} = \frac{\lambda p_i n_{i,2}}{N_i}(1-q_i)\beta \ . \tag{8}$$

The other possible transitions convert an edge of category 0, 1 or 2 into an edge of category 3. This happens when a query writes after receiving an incorrect answer to at least one of its reads. Denoting the corresponding instantaneous transition rates by $a_{i,j,3}$, for $j = 0, 1, 2$, we have

$$a_{i,j,3} = \frac{\lambda p_i n_{i,j}}{N_i}(1-\beta) \ . \tag{9}$$

Using these transition rates, one can simulate the process of corrupting the database and obtain both point estimates and confidence intervals for the average time to corruption, U. However, the systems of practical interest tend to be large, and such simulations take a long time to run. It is therefore desirable to develop an analytical solution that is both efficient to implement and provides accurate estimates for U. That is our next task.

3 Fluid approximation

Instead of describing the system state by integer-valued functions specifying numbers of edges of various types and categories, it is convenient to use continuous fluids of those types and categories. So now $n_{i,j}(t)$ is a real-valued function indicating the amount of fluid present at time t in a 'bucket' of type i and category j ($i = 1, 2, \ldots, T$; $j = 0, 1, 2, 3$). The total amounts of different types, and the initial states, are the same as before.

Fluids flow out of, and into buckets, at rates consistent with the transition rates described in the previous section. Thus, the bucket labeled $(i, 0)$ (local of type i and clean) has an outflow at the rate given by (9), and no inflow. This can be expressed by writing

$$n'_{i,0}(t) = -\frac{\lambda p_i(1-\beta)}{N_i} n_{i,0}(t) \ , \tag{10}$$

where β is given by (3) and (2).

The bucket labeled $(i, 1)$ (distributed of type i and clean) has two outflows, at rates given by (7) and (9) respectively, and an inflow at rate given by (8). The corresponding equation is,

$$n'_{i,1}(t) = -\frac{\lambda p_i(\beta^2 q_i + 1 - \beta)}{N_i} n_{i,1}(t) + \frac{\lambda p_i \beta(1-q_i)}{N_i} n_{i,2}(t) \ . \tag{11}$$

Similarly, bucket $(i, 2)$ (half-corrupted of type i) has two outflows, at rates given by (8) and (9) respectively, and an inflow at rate given by (7). This implies

$$n'_{i,2}(t) = -\frac{\lambda p_i(\beta(1-q_i) + 1 - \beta)}{N_i} n_{i,2}(t) + \frac{\lambda p_i \beta^2 q_i}{N_i} n_{i,1}(t) \ . \tag{12}$$

Finally, bucket $(i,3)$ (corrupted of type i) has three inflows, at rates given by (9), and no outflows. Hence,

$$n'_{i,3}(t) = \frac{\lambda p_i(1-\beta)}{N_i}[n_{i,0}(t) + n_{i,1}(t) + n_{i,2}(t)] \ . \tag{13}$$

The object is to determine the value U such that

$$\sum_{i=1}^{T} n_{i,3}(U) = \gamma N \ . \tag{14}$$

Unfortunately, the above differential equations are coupled in a complicated way. Not only do the unknown functions appear in each others equations, but they also depend on β, which in turn depends on α, which depends on the unknown functions. Moreover, that dependency is non-linear. Consequently, an exact solution for this set of equations does not appear to be feasible. We need another level of approximation.

Denote by $\bar{n}_{i,j}$ the average value of the function $n_{i,j}(t)$ over the interval $(0,U)$:

$$\bar{n}_{i,j} = \frac{1}{U}\int_0^U n_{i,j}(t)dt \ . \tag{15}$$

Replacing, in the right-hand side of (2), all functions by their average values, allows us to treat the probability α as a constant:

$$\alpha = \sum_{i=1}^{T} \frac{p_i}{N_i}[\bar{n}_{i,0} + \bar{n}_{i,1} + \frac{1}{2}\bar{n}_{i,2}] \ . \tag{16}$$

Then the probability β will also be a constant. Also, where one unknown function appears in the differential equation of another, replace the former by its average value. The resulting equations are linear, with constant coefficients, and are easily solvable. The solution of (10), which involves only $n_{i,0}(t)$, becomes

$$n_{i,0}(t) = (1-f)N_i e^{-a_{i,0}t} \ , \tag{17}$$

where $a_{i,0} = \lambda p_i(1-\beta)/N_i$.

In equation (11), $n_{i,2}(t)$ is replaced by $\bar{n}_{i,2}$. The solution is then

$$n_{i,1}(t) = \frac{b_{i,1}\bar{n}_{i,2}}{a_{i,1}}[1 - e^{-a_{i,1}t}] + fN_i e^{-a_{i,1}t} \ , \tag{18}$$

where $a_{i,1} = \lambda p_i(\beta^2 q_i + 1 - \beta)/N_i$ and $b_{i,1} = \lambda p_i \beta(1-q_i)/N_i$.

Similarly, in equation (12), $n_{i,1}(t)$ is replaced by $\bar{n}_{i,1}$. This yields

$$n_{i,2}(t) = \frac{b_{i,2}\bar{n}_{i,1}}{a_{i,2}}[1 - e^{-a_{i,2}t}] \ , \tag{19}$$

where $a_{i,2} = \lambda p_i[\beta(1-q_i) + 1 - \beta]/N_i$ and $b_{i,2} = \lambda p_i \beta^2 q_i/N_i$.

Replacing $n_{i,j}(t)$ by $\bar{n}_{i,j}$ in equation (13), makes the right-hand side constant and therefore

$$n_{i,3}(t) = t\frac{\lambda p_i(1-\beta)}{N_i}(\bar{n}_{i,0} + \bar{n}_{i,1} + \bar{n}_{i,2}) \; . \tag{20}$$

Hence, according to (14), the time to corruption U can be estimated as

$$U = \gamma N \left[\sum_{i=1}^{T} \frac{\lambda p_i(1-\beta)}{N_i}(\bar{n}_{i,0} + \bar{n}_{i,1} + \bar{n}_{i,2})\right]^{-1} \; . \tag{21}$$

Integrating (17), (18) and (19) over the interval $(0, U)$ and dividing by U, we obtain the following expressions:

$$\bar{n}_{i,0} = \frac{(1-f)N_i}{a_{i,0}U}[1 - e^{-a_{i,0}U}] \; ; \tag{22}$$

$$\bar{n}_{i,1} = \frac{b_{i,1}\bar{n}_{i,2}}{a_{i,1}} + (fN_i - \frac{b_{i,1}\bar{n}_{i,2}}{a_{i,1}})\frac{1}{a_{i,1}U}[1 - e^{-a_{i,1}U}] \; ; \tag{23}$$

$$\bar{n}_{i,2} = \frac{b_{i,2}\bar{n}_{i,1}}{a_{i,2}}[1 - \frac{1}{a_{i,2}U}(1 - e^{-a_{i,2}U})] \; . \tag{24}$$

This is a set of non-linear simultaneous equations for the averages $\bar{n}_{i,0}$, $\bar{n}_{i,1}$ and $\bar{n}_{i,2}$. They can be solved by consecutive iterations.

Start with some initial estimates for $\bar{n}_{i,j}$; call them $\bar{n}_{i,j}^{(0)}$. Using (16), get an initial estimate for α and hence for β; call those $\alpha^{(0)}$ and $\beta^{(0)}$. Then (21) provides an initial estimate for U, called $U^{(0)}$.

Substituting the initial estimates into the right-hand sides of (22), (23) and (24), yields new values for the averages $\bar{n}_{i,j}$; call them $\bar{n}_{i,j}^{(1)}$. They in turn provide new values, $\alpha^{(1)}$ and $\beta^{(1)}$, and a new estimate, $U^{(1)}$.

In step m of this procedure, the values $\bar{n}_{i,j}^{(m-1)}$, $\beta^{(m-1)}$ and $U^{(m-1)}$ are used to compute $\alpha^{(m)}$, $\beta^{(m)}$, $\bar{n}_{i,j}^{(m)}$ and $U^{(m)}$. The process terminates when the results of two consecutive iterations are sufficiently close to each other.

In effect, the above procedure computes a fixed point for the mapping $\bar{n}_{i,j} \to \bar{n}_{i,j}$ defined by (22), (23) and (24). Such a fixed point exists by Brouwer's theorem [3] because the averages are bounded and the mapping is continuous.

In the next section, the fluid approximation is used to study the behaviour of a reasonably realistic sample database. The accuracy of the approximation is evaluated by comparisons with simulations.

4 Numerical and simulation results

The example database contains five types of edges. Their numbers are: $N_1 = 10^4$, $N_2 = 10^5$, $N_3 = 10^6$, $N_4 = 10^7$ and $N_5 = 10^8$. The corresponding probabilities of access are $p_1 = 0.5$, $p_2 = 0.26$, $p_3 = 0.13$, $p_4 = 0.07$ and $p_5 = 0.04$. The number of read operations per query is distributed geometrically, starting at 2:

$r_k = (1-r)^{k-2}r$, with $r = 0.07$. Thus, on the average, there are just over 15 reads per query.

The time to complete a distributed write operation is assumed constant, equal to 0.005 seconds.

In all types, a fraction 0.3 of the edges are distributed and the rest are local (for an argument in support of this fraction, see [8]). The database starts clean at time 0 and is considered to be corrupted when a fraction $\gamma = 0.1$ of all edges are corrupted.

In Figure 2, the average period until corruption is plotted against the arrival rate of queries, λ. The latter is varied in the range (100,5000) queries per second. The time U is measured in hours.

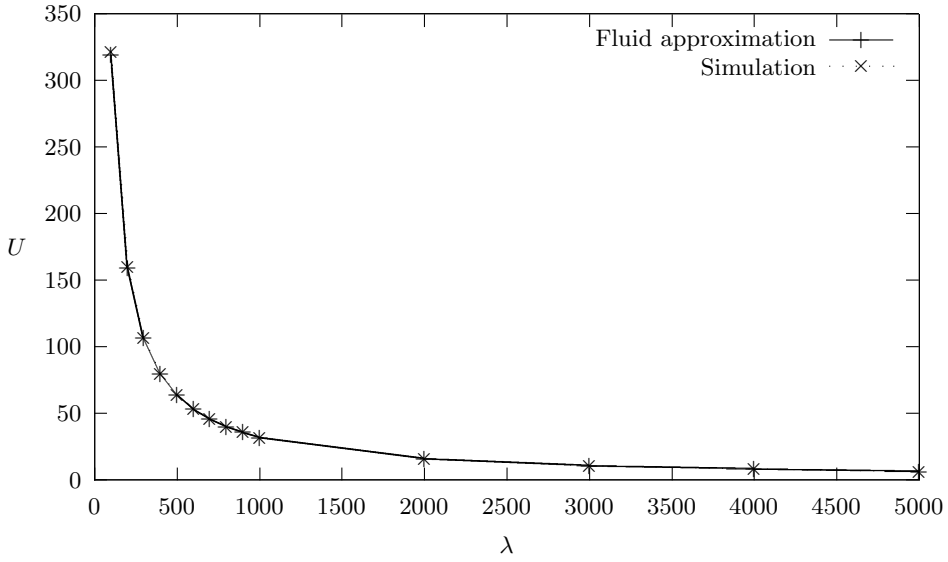

Fig. 2. Corruption time in hours vs. arrival rate/sec.

We observe that U decreases with λ. This was of course to be expected, since a higher arrival rate leads both to higher probability of conflicts, and faster spread of incorrect information. In this database, type 1 forms a relatively small nucleus of edges that are quite likely to be accessed; once they become involved in conflicts, corruption spreads rapidly.

The figure also aims to compare the fluid approximation results with those obtained by simulation. That is, the transition steps governed by the rates (7), (8) and (9) were simulated until an absorption state was reached.

The two plots are practically indistinguishable; the relative differences that exist are smaller than 1%. On the other hand, the fluid approximation plot took

a fraction of a second to compute (each point required fewer than 10 fixed-point iterations), whereas the simulated one took more than half an hour.

The next experiment examines the effect of the average number of read operations per query, $E(K)$, on the time to corruption. The arrival rate is fixed at $\lambda = 500$ queries per second. The other parameters are as in Figure 2.

For the purpose of this evaluation, the requirement that there should be at least two reads per query has been dropped. The random variable K has the normal geometric distribution with parameter r: $r_k = (1-r)^{k-1}r$, $k = 1, 2, \ldots$. In Figure 3, r decreases from 0.99 to 0.02, which means that $E(K) = 1/r$ increases from 1.01 to 50. The time to corruption, U, is again measured in hours.

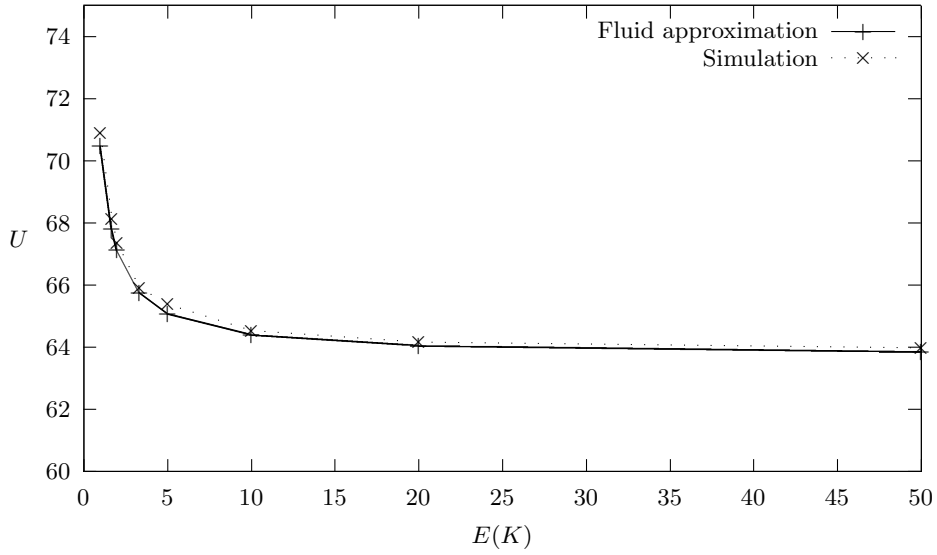

Fig. 3. Corruption time in hours vs. average number of reads

As expected, the more edges are read by queries, the higher the probability of reading a corrupted edge, and hence the shorter the time to a corrupted database. Less obvious, however, is the observation that the resulting decrease in U is highly non-linear. Indeed, increasing $E(K)$ beyond 10 almost ceases to make a difference. We see roughly the same U, whether there are 10 or 50 reads per query.

The accuracy of the fluid approximation is again very good over the entire range of $E(K)$.

It may also be of interest to examine the effect of the fraction of distributed edges, f, on the interval U. In Figure 4, that fraction is varied between $f = 0.1$ and $f = 1$. The arrival rate is fixed at $\lambda = 100$ (in order to prolong the time

to corruption), and the number of reads per query is distributed geometrically starting with 2, with parameter $r = 0.07$ and mean just over 15.

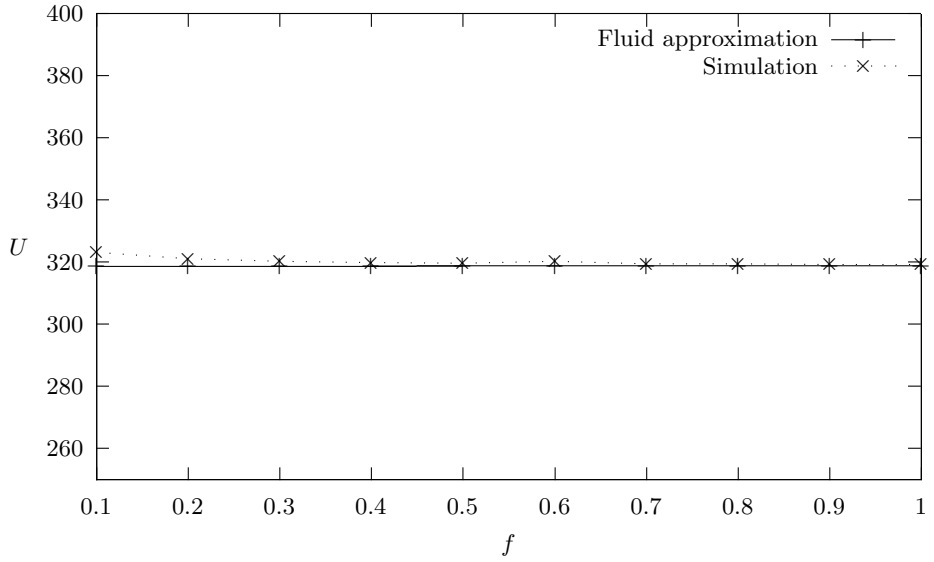

Fig. 4. Varying fraction of distributed edges

The fluid approximation plot is flat. This is not entirely surprising, since the expression for the probability α involves only the sums $\bar{n}_{i,0} + \bar{n}_{i,1}$, and not the individual averages. Moreover, local edges are just as easily corrupted by incorrect reads as distributed ones.

The simulation agrees with the approximation for most of the range, but begins to diverge from it when $f = 0.1$. It seems that when the fraction of distributed edges is very small, the accuracy of the fluid approximation diminishes.

In the final experiment, the parameter that is varied is the fraction, γ, of edges that should become corrupted before the database is considered to be corrupted. The arrival rate is fixed at $\lambda = 500$, and all other parameters are as in Figure 2.

Figure 5 shows how the time to corruption grows when the definition of corruption becomes more demanding. The plot is a convex curve, which is slightly counter-intuitive. One might guess that the more edges are corrupted, the faster even more edges would be corrupted. That would produce a concave curve. In fact the opposite is observed. The likely explanation is that the fewer the remaining clean edges, the longer it takes for a random access to hit one and corrupt it.

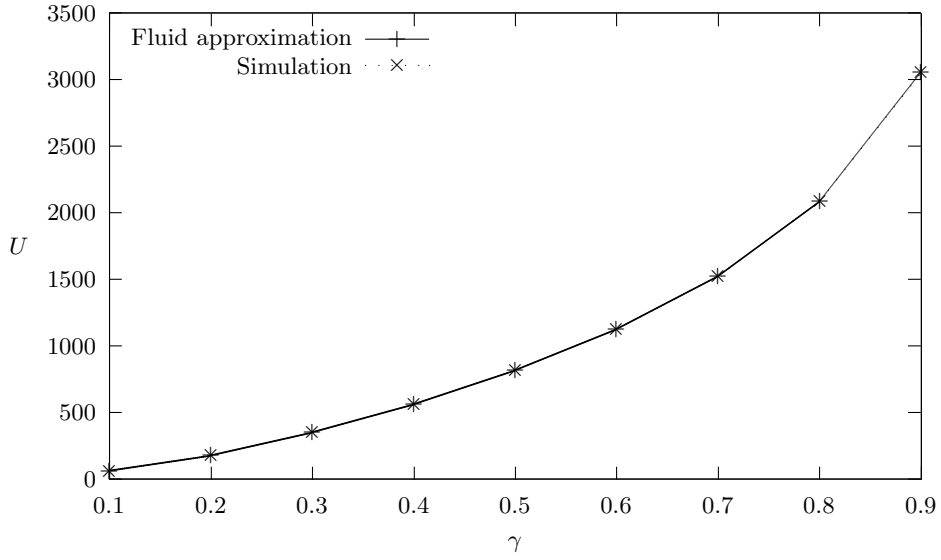

Fig. 5. Varying fraction of corrupted edges

The fluid approximation estimates are almost indistinguishable from the simulation ones, while being several orders of magnitude faster to compute.

5 Conclusions

The problem that we have addressed — to construct and solve a quantitative model of database deterioration — is of considerable practical importance. The fluid approximation that has been developed is fast and provides accurate estimates of the time to corruption. The only area where there is a suggestion of inaccuracy is when the fraction of distributed edges is very small.

It may be possible to improve the approximation so that it can handle more extreme parameter values. One idea would be to break the interval $(0, U)$ into smaller portions and apply the fixed-point iterations consecutively to each portion. This, and other possible extensions would be a suitable topic for future research.

References

1. P. Bailis and A. Ghodsi, "Eventual Consistency Today: Limitations, Extensions, and Beyond", Queue, 11, 3, pp. 20-32, 2013.
2. E.A. Brewer, "Towards robust distributed systems", Procs. 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, 1619, 2000.

3. L.E.J. Brouwer, “Uber Abbildungen von Mannigfaltigkeiten”, Mathematische Annalen, 71, pp. 97-115, 1911.
4. http://cassandra.apache.org/
5. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, “Dynamo: Amazon’s Highly Available Key-value Store”, SIGOPS Oper. Syst. Rev., 41, 6, pp. 205-220, 2007.
6. S. Gilbert and N. Lynch, “Brewers conjecture and the feasibility of consistent, available, partition-tolerant Web services”, ACM SIGACT News 33, 2 (2002).
7. M.P. Herlihi and J.M. Wing, “Linearizability: A Correctness Condition for Concurrent Objects”, ACM TOPLAS, 12, 3, pp. 463-492, 1990.
8. J. Huang and D.J. Abadi, “Leopard: lightweight edge-oriented partitioning and replication for dynamic graphs”, Proceedings VLDB Endowment, 9, 7, pp. 540-551, 2016.
9. I. Robinson, J. Webber and E. Eifrem, *Graph Databases, New Opportunities for Connected Data*, O’Reilly Media, ISBN 978-1491930892, 2015.
10. W. Vogels, “Eventually Consistent”, Comm. ACM, 52,1, pp. 40-44, 2009.