Accurate Performance Modeling and Guidance to the Adoption of an Inconsistency Detection Framework

Yijun Lu[1], Xueming Li[2,1], and Hong Jiang[1]
[1]*Department of Computer Science and Engineering, University of Nebraska-Lincoln*
{yijlu, jiang}@cse.unl.edu
[2]*School of Computer Science and Engineering, Chongqing University, China*
xli@cse.unl.edu

Abstract

With the increased popularity of replica-based services in distributed systems such as the Grid, consistency control among replicas becomes more and more important. To this end, IDF (Inconsistency Detection Framework), a two-layered overlay-based architecture, has been proposed as a new way to solve this problem—instead of enforcing a predefined consistency control protocol, IDF detects inconsistency in a timely manner when it occurs and resolves it based on applications' semantics.

This paper presents a comprehensive analytical study of the two-layer detection mechanism. We develop an analytical model to characterize IDF and, based on this model, evaluating the successful rate of inconsistency detection within the top layer, which directly impacts the performance of IDF. We also derive a unified formula to model the performance of IDF with regard to a wide range of applications. Based on the modeling results, we illustrate how practitioners can use these quantitative insights to tune the IDF parameters for specific applications.

1. Introduction

With the increased popularity of Internet-scale distributed systems, such as the Grid and wide-area e-business applications, replica-based systems have gained momentum. With multiple replicas, the system can improve both the availability—a user can access a nearby copy—and scalability—there is no bottleneck and single point of failure [10]. However, with the presence of multiple replicas, consistency control becomes an important issue. Conventionally, the consistency problem is solved as follows: before the system starts to run, the administrator deploys a pre-defined consistency protocol, such as an optimistic protocol [2] or strong consistency protocol [1]. However, in an environment where multiple applications with different consistency requirements run concurrently, this simple approach is not adequately versatile: sometimes a pre-defined protocol is insufficient, while at other times it can be overkill.

IDF (Inconsistency Detection Framework) provides an alternative [4, 5, 6]. Instead of deploying a pre-defined consistency protocol, IDF lets the system run and detects any inconsistency of the system in a timely manner. When an inconsistency is detected, IDF resolves the inconsistency based on applications' semantics: it neglects the inconsistency if the consistency level is still acceptable and resolves the inconsistency otherwise. In IDF, timely inconsistency detection is the key because it ensures the system's performance, *i.e.*, the response time [5]. Inspired by the observation that a small number of active participants cause most inconsistencies, IDF uses a two-layer structure in which the top layer is formed by the current active writers and the bottom layer contains the rest of participants. This two-layer detection framework captures most inconsistencies in the top layer due to its ability to identify and include active writers, and hence the efficiency of inconsistency detection is greatly improved due to the relatively small size of the top layer.

While our previous research has shown that IDF achieves the design goal with minimal bandwidth cost [5], it is still not clear how much inconsistencies can be captured in the top layer. The importance of this measure lies in the fact that the usefulness of the top-layer depends on whether or not the majority of inconsistencies can be captured there. Also, it is not clear about how a practitioner can adjust or optimize the parameter settings of IDF to suit specific applications when adopting IDF.

As a continuous work to our previous ones, we attempt to answer these two questions in this paper in three steps. First, we develop an analytical model to characterize the main principles of IDF. Second, we accurately derive the successful rate of inconsistency detection by the top layer in all cases. Third and finally, we derive a unified formula to relate these success rates to specific applications.

Beyond validating the design of IDF, the analytical modeling of IDF presented in this paper can benefit research in other areas as well. For example, with more and more applications moving to run on the Internet, a viable way to improve the response time for an application is to use layered structure, similar to the two-layer structure of IDF, and to capture most activities on a relatively small top layer, instead of consulting all participating nodes. Example applications include Peer-to-Peer searching [3] and service discovery [8]. While their application background is different from that for IDF, a similar modeling process to the one presented in this paper can potentially be applied to those research areas as well to quantitatively show the benefit of the layered structure, which is used by both their systems and IDF.

The rest of the paper is organized as follows. Section 2 presents an overview of IDF. Then Section 3 develops the analytical model and analyzes the success rate of the top-layer detection by considering all possible scenarios. Section 4 derives a unified formula to explore the implications of IDF to particular applications and offers insight into possible ways to fine tune IDF. Finally, Section 5 concludes this paper.

2. An Overview of IDF

A logical diagram of IDF is shown in Figure 1. In this framework, multiple applications share data and services through the support of the Internet-scale middleware and inconsistencies among them are detected by the detector. Upon detection, the detector consults with the inconsistency-level monitor (step 1 and step 2) that reflects the consistency requirements of specific applications before any reaction is initiated. Based on applications' semantics, if the inconsistency is tolerable, the detector does not react; otherwise, the detector informs the inconsistency resolution module to resolve this inconsistency (step 3).

The basic idea of timely inconsistency detection is to build a two-layer overlay on top of the underlying network based on nodes' updating history. The top layer consists active writers based on nodes' updating history and is referred as "temperature overlay". The bottom layer contains the rest of the system and is used as a backup and only triggered when the top layer does not find any inconsistency. In the top layer, each node tracks its own updating history and exchanges this information with others periodically. When a node commits an update, this update is propagated in the

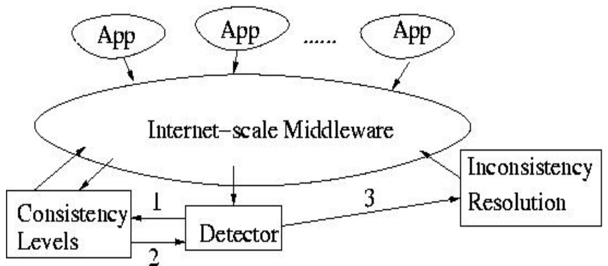


Figure 1. Architecture of the Inconsistency Detection Framework

temperature overlay and the nodes that update this file most frequently are visited first. For detailed description of the protocol, please refer to [4, 5].

2.1. How to analyze IDF?

We analyze IDF from two aspects. First, we measure the success rate that an inconsistency can be detected by the top layer, thus avoiding triggering the bottom layer that is much slower. If the success rate is high, IDF can use top layer to detect most inconsistencies in a timely manner, which translates into better performance—faster detection.

Second, we measure the impact of different applications. This is important because different applications may exhibit difference updating patterns as well as user distributions, thus it is highly likely that, for different applications, the performance of IDF can be different. To give practitioners better understanding of how to best benefit their particular applications from adopting IDF, we need to put the performance of IDF in the context of a wide range of applications.

3. The analytical model

3.1 Assumptions and definitions

The goal of this analysis is to derive the success rate of the top-layer inconsistency detection. In this analysis, we assume that there are n nodes in this system and, after a warm-up period, the top layer associated with a given file f has been formed. This can be visually represented by Figure 2. A copy of file f exists in every writer of it.

For the purpose of analysis and simplicity, we assume that there are two concurrent writers A and B for file f, and each writer can be from either top layer or bottom layer. While there can potentially be more than two concurrent writers, the analysis can be simplified to the case of two writers without the loss of generality because, as far as inconsistency detection is concerned, it is a matter of two writers—the writer that triggers the detection in the first place and whether its inconsistency (in the form conflicting updates) is detected when its update conflicts with that from the first concurrent writer, thus all other concurrent (but later) writers do not matter in this case.

Now let us first define some terms and parameters that we will use throughout the rest of the paper, starting with the following seven events.

[1] E: IDF fails to detect the inconsistency between A and B in the top layer.
[2] e_1: writers A and B are both from the top layer.
[3] e_2: one writer comes from the top layer, and the other comes from the bottom layer.
[4] e_3 : A and B are both from the bottom layer.
[5] d_1: IDF fails to detect inconsistency between A and B in the top layer given event e_1.
[6] d_2: IDF fails to detect inconsistency between A and B in the top layer given event e_2.
[7] d_3: IDF fails to detect the inconsistency in the top layer given event e_3.

As a measure of the performance of IDF, we denote the probabilities of several events as follows.

[1] The overall success rate of the top layer detecting the inconsistency is denoted as P_{succ}.
[2] The probability of event E is denoted as P.
[3] The probability of event e_i (i = 1, 2, 3) occurring is denoted by h_i. Obviously, the sum of h_i (i = 1, 2, 3) is 1.
[4] The probability of event d_i (i = 1, 2, 3) occurring is denoted by p_i.

Clearly, P_{succ} is the overall success rate of the top layer detecting an inconsistency. Intuitively, we have

$$P_{succ} = 1 - P = 1 - \sum_{i=1}^{3} h_i p_i \tag{1}$$

3.2. The analysis

In this section, we analyze the performance of IDF by deriving the formulas for p_i (i = 1, 2, 3)—the failure rates—and use these values as an indicator of the performance of IDF (success rate = 1 – failure rate).

3.2.1. Derivation of p_1

Recall that p_1 is the probability of event d_1, which in turn implies that event e_1 happens. When event e_1 happens, concurrent writers A and B are both from top

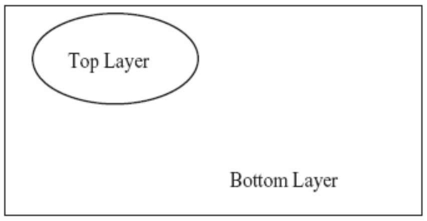


Figure 2. An abstract view of the two-layer system for a given file

layer. According to the system design [5], the top layer is able to detect inconsistency whenever the two writers can find each other. In this case, they can find each other through the top layer, which is visible to both of them. Thus the probability that the system fails to detect the inconsistency is 0. So we have the:

$$p_1 = p(d_1) = 0 \tag{2}$$

3.2.2. Derivation of p_2

Note that p_2 is the probability of event d_2, which in turn implies event e_2. In this case, one of the two writers comes from the top layer and the other is from the bottom layer. Without loss of generality, let us suppose that B is from the top layer while A is from the bottom layer.

Let us assume that there is a set S_{exist} of n_1 nodes in the current top layer, then, from the assumption, we know that B belongs to the set S_{exist} and A is form the bottom layer. After A starts the update operation, its update will be distributed to certain nodes in the network overtime. Without loss of generality, we assume that there is a set SA_{new} of n_2 nodes that have learned that A has become an active writer of the file f by the time both A and B become concurrent writers.

Because there is no particular requirement of these SA_{new} nodes, it is possible that, probabilistically speaking, some nodes in SA_{new} are from S_{exist}. If sets S_{exist} and SA_{new} do not intersect, the top layer will fail to detect this inconsistency because A and B cannot meet each other and the bottom layer has to be triggered (Figure 3(a)). On the other hand, if sets S_{exist} and SA_{new} intersect, this inconsistency can be detected in the top layer because, according to the protocol, as long as A and B can meet each other, the inconsistency can be detected (Figure 3(b)).

Assuming that there are a total of n nodes in the system, we have

$$p_2 = p(d_2) = \frac{C_n^{n_1} C_{n-n_1}^{n_2}}{C_n^{n_1} C_n^{n_2}} = \frac{C_{n-n_1}^{n_2}}{C_n^{n_2}} \tag{3}$$

To calculate p_2, we use the Stirling approximation [7], which states that

$$n! \approx \sqrt{2\pi n}e^{-n}n^n$$

Thus we have

$$p_2 = p(d_2) \approx \sqrt{\frac{(n-n_1)(n-n_2)}{n(n-n_1-n_2)}}\frac{(n-n_1)^{n-n_1}(n-n_2)^{n-n_2}}{n^n(n-n_1-n_2)^{n-n_1-n_2}}$$

3.2.3. Derivation of p_3

The value of p_3 is the probability of event d_3, which in turn implies the occurrence of event e_3,, meaning that both *A* and *B* are from the bottom layer. Following the same analysis in Section 3.2.2, we can assume the existence of three sets S_{exist}, SA_{new} and SB_{new}, because both *A* and *B* are from the bottom layer,

Depending on whether the three sets intersect with each other, there are six cases that need to be considered. The six cases are illustrated in Figure 4 (from 4(a) to 4(f)).

For simplicity of analysis, we use n_1, n_2 and n_3 to denote the sizes of S_{exist}, SA_{new}, and SB_{new}, respectively. According to the IDF protocol, the top layer will fail to detect any inconsistency in cases (*a*), (*b*) and (*c*); and will be able to detect inconsistency in the remaining cases (*d*, *e*, and *f*) where *A* and *B* can meet each other eventually.

Let g_1, g_2 and g_3 denote the probabilities of cases (*a*), (*b*) and (*c*), respectively. Then we have

$$g_1 = \frac{C_n^{n_1}C_{n-n_1}^{n_2}C_{n-n_1-n_2}^{n_3}}{C_n^{n_1}C_n^{n_2}C_n^{n_3}}$$

$$g_2 = \frac{C_n^{n_3}[C_{n-n_3}^{n_1}C_{n-n_3}^{n_2} - C_{n-n_3}^{n_1}C_{n-n_3-n_1}^{n_2}]}{C_n^{n_1}C_n^{n_2}C_n^{n_3}}$$

$$g_3 = \frac{C_n^{n_2}[C_{n-n_2}^{n_1}C_{n-n_2}^{n_3} - C_{n-n_2}^{n_1}C_{n-n_2-n_1}^{n_3}]}{C_n^{n_1}C_n^{n_2}C_n^{n_3}}$$

To simplify the formula, we can assume that n_2 is equal to n_3, which is reasonable for analysis purpose because *A* and *B* have equal status and behave similarly. Then we have

$$g_1 = \frac{C_n^{n_1}C_{n-n_1}^{n_2}C_{n-n_1-n_2}^{n_3}}{C_n^{n_1}C_n^{n_2}C_n^{n_3}} = \frac{C_{n-n_1}^{n_2}C_{n-n_1-n_2}^{n_2}}{(C_n^{n_2})^2}$$

$$g_2 = g_3 = \frac{C_{n-n_2}^{n_1}[C_{n-n_2}^{n_2} - C_{n-n_2-n_1}^{n_2}]}{C_n^{n_1}C_n^{n_2}}$$

Finally, we have

$$p_3 = p(d_3) = g_1 + g_2 + g_3 \qquad (4)$$

4. Application characteristics and the unified formula

In this section, we explore the performance and design implications of IDF to specific applications. In particular, we relate the failure rate of the top-layer detection (the values of p_1, p_2, and p_3) to real applications by deriving a unified formula. We then show how to adjust two parameters in the unified formula to reflect a particular application's user distribution, as well as usage pattern. In this way, a practitioner can adjust certain IDF protocol parameters to potentially optimize the IDF performance for his or her particular applications.

As discussed in Section 3, to derive a unified formula for IDF as a whole, P_{succ}, thus *P*, has to be derived (see formula (1) in Section 3). In turn, we have to derive h_i (i = 1, 2, 3) first (h_i is the probability that event d_i occurs).

4.1. Random and uniform updates

In step 1, we assume that all nodes have an equal probability of updating, thus the probability of a writer coming from the top layer is decided by the relative size of the top layer. For the same reason, the probability of a writer coming from the bottom layer is decided by the relative size of the bottom layer.

Now we can evaluate h_i (i = 1, 2, 3) as follows:

[1] The event d_1 means that the two concurrent writers *A* and *B* are both from the top layer. In this case, we have

$$h_1 = \frac{C_{n_1}^2}{C_n^2}$$

[2] The event d_2 means that one of the two concurrent writers comes from the top layer, while the other is from bottom layer. In this case, we have

$$h_2 = \frac{C_{n_1}^1 C_{n-n_1}^1}{C_n^2}$$

[3] The event d_3 means that the two concurrent writers are both from the bottom layer. In this case, we have

$$h_3 = \frac{C_{n-n_1}^2}{C_n^2}$$

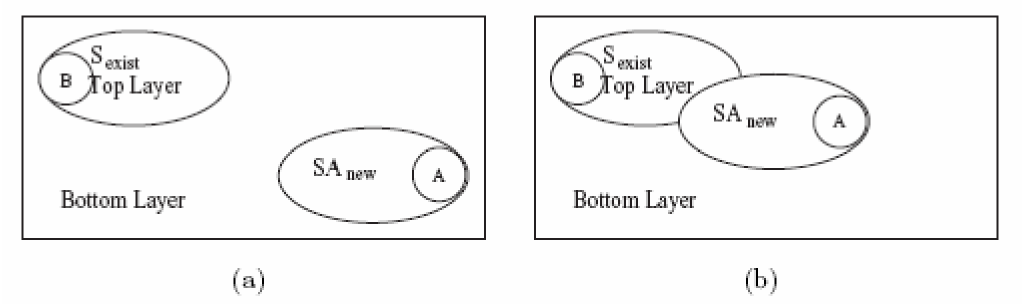
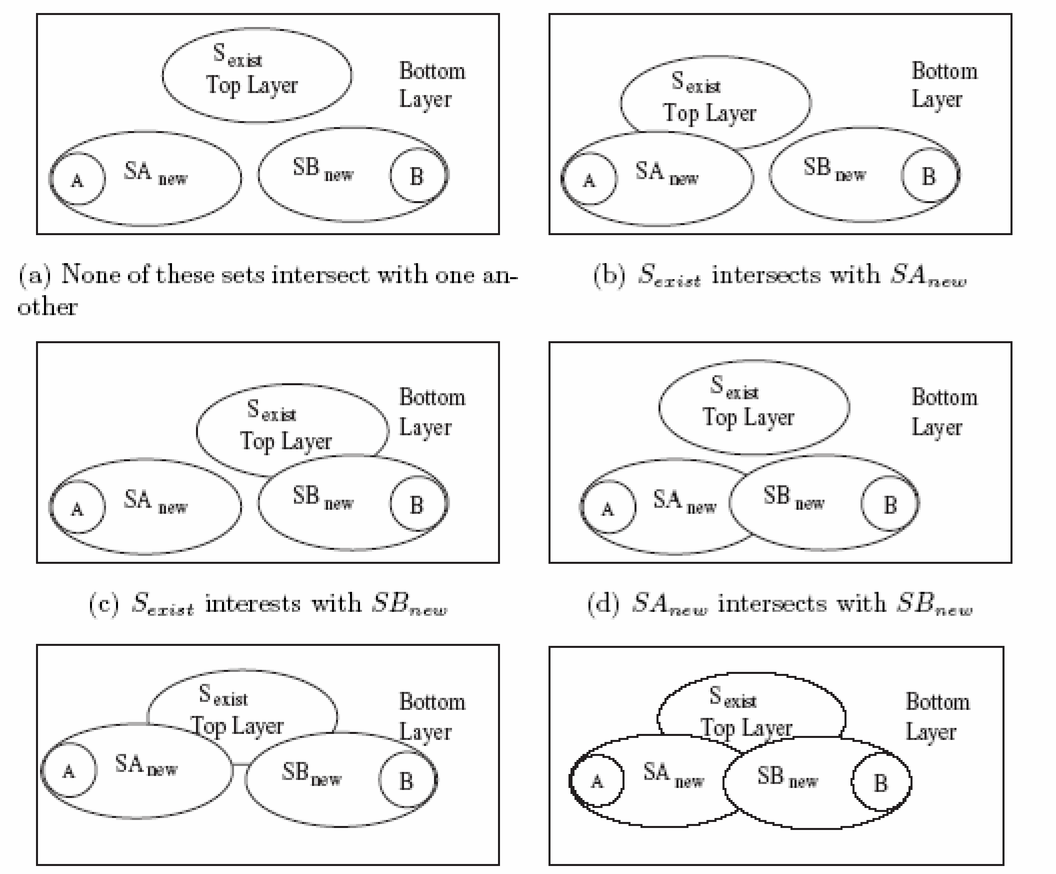


Figure 3. **Two cases given event e_2**



(a) None of these sets intersect with one another

(b) S_{exist} intersects with SA_{new}

(c) S_{exist} interests with SB_{new}

(d) SA_{new} intersects with SB_{new}

(e) Both SA_{new} and SB_{new} intersect with S_{exist}, but SA_{new} and SB_{new} do not intersect with each other

(f) Three sets intersect with one another

Figure 4. **Six cases given event e_3**

4.2. Non-uniform updates between two layers

In this step, we relax the random and uniform assumption made in Step 1 above by assuming that the updating frequency of the nodes in the top layer is F_1 and that of the nodes in the bottom layer is F_2. However, it is implied that within each layer, all nodes are equally likely to issue update requests for a given file.

In practice, if F_1 is larger than F_2, it means that nodes in the top layer have more update requests than the nodes in the bottom layer, which is the case for applications like online gaming in which active gamers keep playing the game for a long period of time (thus they will form a top layer and keep issuing updating requests).

If F_2 is larger than F_1 instead, it means that new updating requests are more likely from bottom layer. This is possible because, while the current top layer reflects the history, it is still possible that the future pattern is different from the history. For example, if there is a forum in which its user base keeps changing and number of requests from new members is larger than that from current members, F_2 would be larger than F_1.

After incorporating the frequencies of updating operations of the top layer and bottom layer nodes, the frequency of event d_1 can be expressed as $F_1^2 C_{n_1}^2$, the frequency of event d_2 as $F_1 F_2 C_{n_1}^1 C_{n-n_1}^1$, and the frequency of the event d_3 as $F_2^2 C_{n-n_1}^2$. Hence, we have

[1] The value of h_1 can be calculated as

$$h_1 = \frac{F_1^2 C_{n_1}^2}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

[2] The value of h_2 can be calculated as

$$h_2 = \frac{F_1 F_2 C_{n_1}^1 C_{n-n_1}^1}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

[3] The value of h_3 can be calculated as

$$h_3 = \frac{F_2^2 C_{n-n_1}^2}{F_1^2 C_{n_1}^2 + F_1 F_2 C_{n_1}^1 C_{n-n_1}^1 + F_2^2 C_{n-n_1}^2}$$

To normalize the values of *F1* and *F2*, We introduce f_1 and f_2, which are defined as

$$f_1 = \frac{F_1}{F_1 + F_2}, \quad f_2 = \frac{F_2}{F_1 + F_2} = 1 - f_1$$

After the normalization, both f_1 and f_2 are in the range of (0, 1). Then h_1, h_2, and h_3 can be expressed as follows

$$h_1 = \frac{f_1^2 C_{n_1}^2}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

$$h_2 = \frac{f_1 f_2 C_{n_1}^1 C_{n-n_1}^1}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

$$h_3 = \frac{f_2^2 C_{n-n_1}^2}{f_1^2 C_{n_1}^2 + f_1 f_2 C_{n_1}^1 C_{n-n_1}^1 + f_2^2 C_{n-n_1}^2}$$

4.3. The unified formula

We can see that, if f_1 equals to f_2, then the values of h_1, h_2, and h_3 are exactly the same as those derived in step 1. In short, the formulas of step 1 are special cases of the formulas we derived here.

Now we simplify these expressions and take two important aspects of real applications into consideration. The two aspects are based on the observation from inside IDF and outside IDF, respectively. The meanings of the two aspects, the justification of their inclusion, and their relationships with the parameter of the formula for h_i (i = 1, 2, 3) are summarized as follows.

- **User distribution**: This aspect characterizes the distribution of the active writers, *i.e.* how many users are active writers in the system. The inclusion of this aspect is due to the inherent two-layer architecture of IDF. As the single important characteristic of IDF, it defines the performance of IDF from inside. User distribution directly affects the size of top layer (the n_1 value).
- **Usage pattern**: This aspect characterizes where the new updates are most likely coming from. For example, if the updating operations are highly concentrated on a small group of dedicated users, then most new updates would come from the top layer; otherwise, more updates would come from the bottom layer. This aspect is from outside IDF and its inclusion is due to the importance of usage patterns in data sharing systems. In e-commerce, for example, the usage pattern is the main focus based on which the system performance can be optimized [9]. In the case of IDF, usage pattern determines the value of f_1, as well as f_2 because $f_2 = 1 - f_1$.

To formally represent the two aspects in the formulas, we do the following substitutions. First, we use α to substitute n_1/n and n_1 in the formulas can be then represented by nα. Second, we use β to substitute f_1 and (1 – β) to substitute f_2. Please note that now both α and β are in the range of (0, 1). Given the total number of nodes in the system—the value of n—the performance of IDF for different applications can be obtained by adjusting the α and β values.

Now we can finally represent h_1, h_2, and h_3 as

$$h_1 = \frac{\beta^2 C_{n\alpha}^2}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \quad \text{… (5)}$$

$$h_2 = \frac{\beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \quad \text{… (6)}$$

$$h_3 = \frac{(1-\beta)^2 C_{n-n\alpha}^2}{\beta^2 C_{n\alpha}^2 + \beta(1-\beta) C_{n\alpha}^1 C_{n-n\alpha}^1 + (1-\beta)^2 C_{n-n\alpha}^2} \quad \text{… (7)}$$

Now that we have h_1, h_2, and h_3, we finally get a unified formula to evaluate the performance of IDF based on formula (1) in Section 3.1. This unified formula can reflect the different characteristics of particular applications by adjusting the values of α and β, which are in the range of (0, 1). Based on this unified formula, the performance of IDF in the context of a wide range of applications can be derived. The results and how they can be used by practitioners to tune IDF are discussed below.

4.4. Results and their implications

In this section, we show results from the analytical model and offer insights about how much benefit particular applications can gain from adopting IDF and, more importantly, how a practitioner can tune the IDF's parameters to tailor it for his or her particular needs.

First of all, a set of results based on the formula 1 (p_i (i = 1, 2, 3) are from the formulas 2 to 4 in Section 3.2; h_i (i = 1, 2, 3) are from the formulas 5 to 7 in Section 4.3) we have derived are summarized in Table 1 that lists several important variables including the detection success rate. From this table, we can see that the lowest success rate is 91.4% while in the vast majority cases the rate is more than 98%, which we believe is very encouraging because it states that the top layer is indeed able to detect most inconsistencies.

Now we take a closer look of the impact of the parameters. First, intuitively, the larger the value of α and n_2 are, the high success rate of detection should be achieved. The results in Table 1 are consistent with this. Second, the impact of the value of β is more complex because it increases the possibility of h_1 and h_2, but decreases that of h_3. From Table 1, we can see that the success rate decreases when β increases. However, at some point, especially when β is close to 1, success rate of detection should start to increase because, when β is close to 1, most writers are from the top layer and the success rate of those detections would be 1. To validate this hypothesis, we collect another set of results as shown in Table 2 that clearly shows that the success rate start to increase when β reaches 0.91.

Now, we investigate how the parameters of IDF can affect the system performance for different applications. In the four parameters we discussed—n_1, n_2, α, and β—n_1 and α correlate ($n_1 = n*α$). Besides, β is the relative activity of top layer nodes comparing with bottom layer nodes and is entirely application-dependent. The larger the value of β is, the more updates are from the top layer nodes. However, α is adjustable because the protocol can lower the threshold of the required temperature for the top layer participants, which in turn can increase the size of the top layer; n_2 is also adjustable because it is a parameter in IDF. The larger the value of α and n_2 are, the higher the success rate is.

For all the discussions, we choose two values of β, 0.8 and 0.2, and investigate the impact of the two values, α and n_2, as follows. We first fix α and β, then calculate P_{succ} based on different values of n_2. The result is shown in Figure 5(a). Two (α, β) pairs and four different n_2 values are used in this calculation.

Then we fix n_2 and β and calculate P_{succ} based on different values of n_1. The result is shown in Figure 5(b). Two (n_2, β) pairs and four different α values are used in this calculation.

From the two figures, we can see that the success rate keeps increasing with the size of n_1 and n_2 values. This makes sense because, with the increased top layer size (n_1) and the limited broadcasting size of a new writer (n_2), there are more chances for concurrent writers to meet. But this comes at a cost as well: slower detection delay due to the larger top layer size (in the case of increased size of $α$) and increased network bandwidth overhead (in the case of increased size of n_2).

Based on this information, the practitioners can fine tune the parameters as follows. If the system has enough bandwidth available, they can increase the value of n_2; if the response time of IDF is better than the required value, it is also possible that they can increase the value of α. While simple, we believe that this suggestion can at least help practitioners to tune IDF in the best way to achieve the best performance of their particular applications.

5. Conclusions

In this paper, we first developed an analytical model to characterize IDF and calculated the success rate of the top-layer detecting an inconsistency (reflected through the failure rate) that directly impacts the performance of IDF. Then, we derived a unified formula to model a wide range of applications and gives guidance to determine how much performance gain particular applications can achieve from adopting IDF and how practitioners can fine tune IDF's parameters to best suit their applications' needs. We believe that this analytical study provides a solid ground work for understanding the concept of IDF and the guidance we provided can help practitioners adopt IDF better.

	α	$n_1 = n * \alpha$	n_2	β	p_2	p_3	h_1	h_2	h_3	P	P_{succ}
0	0.02	20	50	0.2	35.487	4.252	0.002	1.011	98.986	4.568	95.432
1				0.5	35.487	4.252	0.038	3.924	96.038	5.476	94.524
2				0.8	35.487	4.252	0.542	13.971	85.487	8.593	91.417
3			80	0.2	18.556	0.032	0.002	1.011	98.986	0.220	99.780
4				0.5	18.556	0.032	0.038	3.924	96.038	0.759	99.241
5				0.8	18.556	0.032	0.542	13.971	85.487	2.620	97.380
6	0.05	50	50	0.2	7.198	1.004	0.017	2.566	97.417	1.163	98.837
7				0.5	7.198	1.004	0.245	9.510	90.245	1.590	98.410
8				0.8	7.198	1.004	2.968	28.772	68.260	2.756	97.244
9			80	0.2	1.385	0.003	0.017	2.566	97.417	0.038	99.962
10				0.5	1.385	0.003	0.245	9.510	90.245	0.134	99.864
11				0.8	1.385	0.003	2.968	28.772	68.260	0.400	99.600
12	0.1	100	50	0.2	0.448	0.064	0.072	5.265	94.663	0.084	99.916
13				0.5	0.448	0.064	0.991	18.018	80.991	0.133	99.867
14				0.8	0.448	0.064	9.387	42.667	47.947	0.222	99.778
15			80	0.2	0.015	0.000	0.072	5.265	94.663	0.001	99.999
16				0.5	0.015	0.000	0.991	18.018	80.991	0.003	99.997
17				0.8	0.015	0.000	9.387	42.667	47.947	0.006	99.994

Table 1: Success rate (in percentage) when *n* is 1000 with 18 sets of parameter values

	α	$n_1 = n * \alpha$	n_2	β	P_{succ}
1	0.1	100	50	0.8	99.778
2				0.9	99.759
3				0.91	99.762
4				0.95	99.871

Table 2: Investigate the impact of

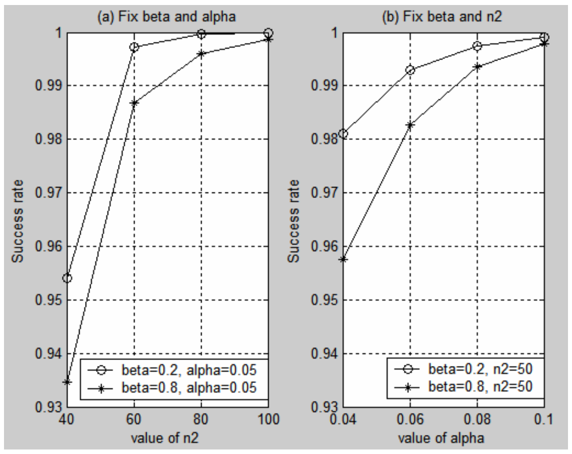


Figure 5. Tuning IDF parameters with n = 1000

References

[1] U. Cetintemel, P. J. Keleher, B. Bhattacharjee, and M. J. Franklin, Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weakly-Connected Environments, IEEE Trans. on Computers, 52(7), 2003.

[2] James J. Kistler and M. Satyanarayanan, Disconnected Operation in the Coda File System, ACM Transactions on Computer Systems, 10(1) pp. 3-25, February 1992.

[3] R. D. Larsen and N.O. Bouvin, HyperPeer: Searching for Resemblance in a P2P Network, in Proc. of 15th ACM Conference on Hypertext and Hypermedia, Santa Cruz, CA, USA, 2004. pp. 268-269.

[4] Y. Lu and H. Jiang, A Framework for Efficient Inconsistency Detection in a Grid and Internet-Scale Distributed Environment, In Proc. of HPDC-14, NC, July 24-27, 2005. pp. 318-319.

[5] Y. Lu, H. Jiang, and Dan Feng, An Efficient, Low-Cost Inconsistency Detection Framework for Data and Service Sharing in an Internet-Scale System, In Proc. of IEEE ICEBE, Beijing, China, Oct. 18-20, 2005. pp. 373-380.

[6] Y. Lu, Y. Lu, and H. Jiang, IDEA: An Infrastructure for Detection-based Adaptive Consistency Control in Replicated Services, In Proc. of 16th HPDC, Monterey, CA, June 2007. ACM Press, pp. 223-224.

[7] Dan Romik, Stirling's Approximation for n!: The Ultimate Short Proof? The American Mathematical Monthly, Vol. 107, No. 6, pp. 556-557.

[8] J. Tyan and Q.H. Mahmoud, A Comprehensive Service Discovery Solution for Mobile Ad Hoc Networks, Mobile Networks and Applications, Vol. 10, issue 4, August, 2005. pp. 423-434.

[9] D. VanderMeer, K. Dutta, and A. Datta, Enabling Scalable Online Personalization on the Web, In Proc. of 2nd ACM Conference on Electronic Commerce, Minneapolis, Minnesota, Oct. 2000. pp. 185-196.

[10] H. Yu and A. Vahdat, Design and Evaluation of a Continuous Consistency Model for Replicated Services, In. Proc. OSDI 2000.