Towards Byzantine Resilient Directories

Ali Shoker
University of Toulouse III, IRIT Lab.
Toulouse, France
ali.shoker@irit.fr

Jean-Paul Bahsoun
University of Toulouse III, IRIT Lab.
Toulouse, France
bahsoun@irit.fr

***Abstract*—Notable *Byzantine Fault Tolerant* protocols have been designed so far. These protocols are often evaluated on simple benchmarks, and few times on *NFS* systems. On the contrary, studies that addressed the behavior of BFT on large *back-ends*, like *Directories*, are few. We believe that studying such systems is crucial for practice community due to their popularity. In this paper, we integrate BFT with *OpenLDAP Directory*. We introduce the design of the integrated system, that we call *BFT-LDAP*. Then, we study its behavior accompanied with some useful observations. In addition, we discuss the cost overhead of this integration. Our approach ensures that OpenLDAP legacy code remains completely intact, and that the integration with BFT is straightforward using *API*s. Moreover, we convey that the additional performance cost of BFT-LDAP is negligible as compared to that of stand-alone OpenLDAP. We conducted our experiments on *Emulab*. The experiments indicate that the performance discrepancy of BFT-LDAP is negligible whenever different state-of-the-art BFT protocols are used. Other experiments demonstrate that a little sacrifice in throughput (less than 10%) is needed in order to leverage the resiliency of OpenLDAP against *Byzantine* faults (i.e., through applying BFT).**

***Keywords*-distributed systems; Byzantine fault tolerance; directories; OpenLDAP;**

I. Introduction

Byzantine fault tolerance [1] (BFT) is a replication approach used to leverage the resiliency of state-machines against *Byzantine* (arbitrary) faults. A BFT protocol is used to manipulate the communication among system replicas under partial synchrony [2]. A BFT protocol requires at least $3f+1$ replicas to ensure consistency among system replicas, where at most f replicas can be Byzantine [1], [3], [4].

During the last decade, research community introduced several BFT solutions [3], [5], [6], [7], [8], [9] that are fairly convincing. Thus, we believe that BFT technology has become mature enough to be deployed in prctice. Nevertheless, perhaps for comparison reasons, most of these protocols were evaluated on simple *increment-value* benchmarks (or on *NFS* systems, sometimes); and few studies analyzed the behavior of BFT on larger *back-end* systems like Databases, Directories, Web applications, etc.

Directories [10] are intended to store and organize data in a repository fashion. They may hold critical information such as governmental and organizational procedures like citizenship, resource repositories, etc. This sensitive data demands high reliability requirements, especially since Directory services are designed to be *open*. *US DISA* highlighted the problem by stating in [11] that: if the confidentiality, integrity, or availability of the related directory service is compromised, it is likely that the security of a dependent server or workstation is also compromised. In [12] and [13], the authors studied six possible attacking threats on *Active and Virtual Directories*, respectively (e.g., spoofing, tampering, repudiation, etc).

Therefore, we believe that applying BFT technology to Directories is crucial to yield high resiliency against arbitrary and malicious faults.

In this paper, we integrate BFT with *OpenLDAP* [14] as a Directory application. We call the integrated system: BFT-LDAP. Four BFT protocols are considered: PBFT [3], Zyzzyva [5], Chain [8], and Quorum [8]. The paper presents the architecture of BFT-LDAP and studies its performance. Three main points we observed: (1) applying BFT technology on services that provide a well-defined API, like OpenLDAP, is cheap and straightforward; (2) the performance overhead of BFT-LDAP (due to replication) is fairly small; and (3) the performance of distinct BFT protocols is very close on applications that exhibit a high processing time (like OpenLDAP).

BFT-LDAP is designed as follows: In addition to the OpenLDAP server and the client application, three additional modules are built to integrate the BFT library onto OpenLDAP: BFT Client Proxy, BFT Server Proxy, and LDAP Access Unit. The BFT client and the client application form the client tier, whereas, the other modules form the server tier, and hence they are installed on each replica (i.e., on the $3f+1$ replicas). The client application sends the requests to the BFT client proxy which forwards the request to the BFT server proxies after performing necessary message digests and encryptions. This communication phase takes place according to the specific message patterns of each BFT protocol. The BFT server proxy calls the LDAP Access Unit that interfaces the OpenLDAP server. After executing the operation on each OpenLDAP server, the reply is sent back through the same modules, however in the opposite sense, again considering the message patterns of the BFT protocol being used.

Our experience showed that integrating BFT with OpenLDAP is straightforward through using its API. We integrated our system using *C/C++* code. The integration ensured that OpenLDAP legacy code is kept completely intact. Only 120 lines were updated/added on the BFT side, and 245 lines were added to implement the mediator between BFT and OpenLDAP, i.e., the *LDAP Access Unit*. The integration took less than one month to be accomplished by one engineer including: understanding OpenLDAP business, system design, implementation, configuration, and testing. We implemented the main directory operations: search, add, modify, and delete.

Moreover, in practice, replication induces additional costs on deployed services. Our observation on BFT-LDAP integration shows that the performance cost overhead is fairly small and tolerable. Our experiments indicate that the throughput of OpenLDAP drops by, only, less than 10% upon applying BFT replication. Thus, service providers would not have to sacrifice a lot while using BFT. Notice that BFT-LDAP requires $3f$+1 replicas (assuming f failed replicas) instead of one replica in a stand-alone OpenLDAP configuration, however, this is not a major issue since: servers are cheap nowadays, and most service providers recommend quality of service over cost.

Furthermore we show that the performance differences among state-of-the-art BFT protocols are negligible in BFT-LDAP, though it is not the case in previous studies like [5], [8], etc. This is referred to the large execution time of *Read-/Update* operations of OpenLDAP (more than $200\mu s$). We believe that other applications with large processing time, like OpenLDAP, have the same impact on the performance of BFT protocols.

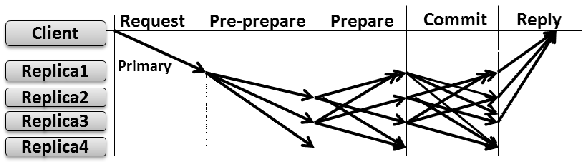
We conducted our experiments on *Emulab* [15] using Xeon machines with *Ubuntu* OS installed. The experiments explore the throughput and latency of BFT-LDAP. We do not study BFT-LDAP in presence of failures since *Transactions* in OpenLDAP API library are designed [16], but not yet implemented. Hence, we defer this to future work.

The rest of the paper is organized as follows. A background on BFT and OpenLDAP is presented in Section II. Section III describes the design of BFT-LDAP. We present an evaluation and some observations in Section V. Then, we reveal related work in Section VI, and we conclude the paper in Section VII.

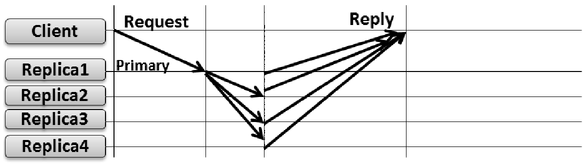
II. Background

A. Byzantine Fault Tolerance

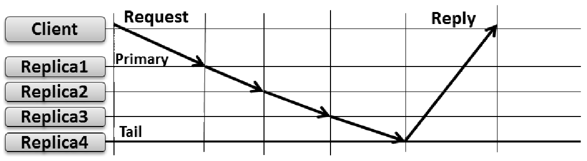
1) Overview: Byzantine fault tolerant protocols are replication-based solutions designed to provide resiliency against *Byzantine* (or arbitrary) faults in state machine systems [17]. BFT protocols ensure correct asynchronous service facing human attacks and software misbehaviors; whereas, traditional fail-stop systems could only resolve crash faults.



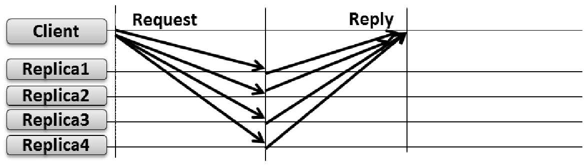
(a) PBFT



(b) Zyzyyva



(c) Chain



(d) Quorum

Figure 1. Message patterns of the state-of-the-art BFT protocols; for $f = 1$.

A BFT protocol uses consensus to ensure one-copy semantics when up to one third of the replicas is Byzantine [4], [1], [3]. Any BFT protocol should maintain two properties: (1) requests are executed in the same order on all replicas, hence preventing individual replica updates, and (2) correct responses are eventually delivered to correct clients.

In BFT protocols, clients issue requests to be executed on replicas; these requests are disseminated among replicas (i.e., agreement) and are totally ordered, usually, by a specific *primary* replica. Then, one or more replicas send back the replies to the client. The client can send another request only if its previous request is completed. Replicas and clients maintain request histories to recover under failure.*Speculative* protocols like [5], [6], [7], [8] launch a recovery phase whenever failures are detected. BFT protocols make use of cryptographic techniques such as *Public Key Cryptography* (PKC) or *Message Authentication Code* (MAC).

2) State-of-the-Art BFT Protocols: ***PBFT*** [3] is the seminal BFT protocol. It is considered the most robust protocol (i.e., it can operate under failure); however, in most cases, it exhibits a lower performance than that of the *speculative* protocols ([5], [6], [7], [8]). A normal request operation is completed in three phases: *pre-prepare*, where the primary proposes a value to all other replicas; *prepare* phase, where the replicas agree on the value and send the acknowledgment to all other replicas; and *commit* phase, where replicas promise to commit the request by broadcasting a *COMMIT* message. Upon receiving $2f$+1 matching COMMIT messages, each replica executes the request and replies to the client; this phase is necessary to detect competing primaries, which might happen during a *view change*. Figure 1(a) shows the message exchange pattern of PBFT. *View change* is a recovery phase required to replace a faulty primary. Measuring performance using benchmarking was first appeared in PBFT [3].

In ***Zyzzyva*** [5], the client sends the request only to the primary replica (figure 1(b)). However, after the remaining replicas receive the request as well as its sequence number from the primary, they immediately execute it (speculatively) and send the reply back to the client. When a faulty replica is detected, the client demands changing the primary. By using speculation, Zyzzyva enhanced the throughput of PBFT significantly in free-failure cases.

The notion of abortability [18] was introduced to reduce the complexity of designing BFT protocols. An abortable BFT protocol (i.e., an *Abstract*) can abort at any time upon request; afterwards, no client request will be serviced by the aborted protocol. A client can initiate abort if the currently running instance of BFT cannot safely progress anymore (e.g., because of contention or a failure), or the performance is unsatisfactory (e.g., because of a change in the payloads). The client then collects the *abort history* that is derived from the local histories of replicas. Then, the client is in charge of launching another backup protocol (called *Backup*), and initializing it with the abort history. Among the well-known *Abstracts*, we mention: Chain, and Quorum [8]. *Re-Abstract Family* [9] is a $4f$+1 version of Abstract where $3f$+1 replicas are $Active$, and f $Passive$ replicas are used as backup replicas. When failures occur, an Abstract recovers to itself, however, after replacing the faulty replicas with correct ones from the f Passive replicas.

Chain [8] is an abortable speculative protocol. It uses a primary replica (called *head*) to receive and order requests (figure 1(c)). All other replicas are ordered in a chain fashion, and each one forwards the request to its successor. The last replica in the chain, i.e., the tail, sends the reply back to the client.Although this technique increases the end-to-end delay, the throughput improves as the number of MAC operations by each replica is close to one, i.e. the theoretical lower bound.

Quorum [18] is also another abortable BFT protocol. In contention-free environments, Quorum has the lowest latency among state of the art protocols. The message communication pattern of Quorum is depicted in (Figure 1(d)). Quorum loses its performance under contention. In the case of contention or a Byzantine behavior, Quorum aborts and recovers to a *Backup* [18] protocol similar to PBFT.

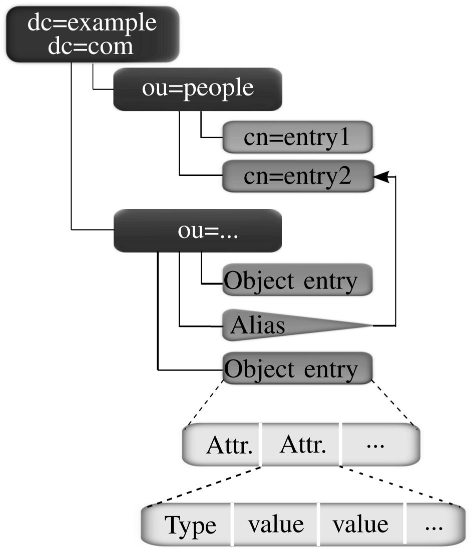


Figure 2. The structure of the *Directory Information Tree*.

B. OpenLDAP

1) Overview: LDAP [19] (*Lightweight Directory Access Protocol*) is a standard messaging protocol used to access (Read/Write) *Directory* information; where a *Directory* is a listing of information about objects arranged in some order [20]. The Directory information are organized in a hierarchical tree structure called *Directory Information Tree*(DIT). Figure 2 presents an example of DIT. The entries of a DIT are usually collections of attributes having values of different types, each entry has a globally-unique *Distinguished Name DN* (e.g., $entry1.people.example.com$).

LDAP technology has grown up fast during the last decade due to its platform-independence, flexibility, and simplicity; especially, whenever a centralized/shared data service is required. Among the real applications of LDAP in industry we reveal: machine and user authentication, e-mail address lookup, group system management, etc [21].

2) OpenLDAP Project: OpenLDAP [14] is an open source project [22] that implements LDAP and provides server and client tools with many features. It supports various platforms, embedded databases, and transactional databases. OpenLDAP provides an API that allows other software to integrate with it smoothly, and hence access most of its features.

OpenLDAP includes a stand-alone server, called *slapd* server, which is a daemon implementation of LDAP composed of: a front-end module and a back-end module. The front-end implements the basic features like: simple SASL authentication and data security, transport layer security (TLS or SSL), access control, data schema, threading, and replication [21]. On the other hand, the back-end includes the storage interfaces and databases. The storage can be in the form of plain text (*back-ldif*), or in other embedded databases like *Oracle Berkeley DB* or HDB (a hierarchical variant of Berkeley databases) [21]. Other transactional DBs, e.g., *Mysql NDB*, are also supported.

To achieve a good performance and availability, OpenLDAP supports many replication schemes; however, they rely on real-time updates and, thus, could not ensure service resiliency whenever synchronization fails or Byzantine faults occur. We do not discuss this type of replication in our paper.

III. Design of BFT-LDAP

A. Model and Assumptions

We adopt the model of BFT protocols [3]. We assume that faulty replicas and clients may behave arbitrarily. An adversary coordinates faulty replicas to compromise the replicated service. However, we assume that the adversary cannot break cryptographic techniques like collision-resistant hashes, encryption, and signatures. System nodes are connected via a network that may fail to deliver, corrupt, delay, or reorder messages. Safety properties hold in any asynchronous environment. Liveness, however, is guaranteed only whenever the system is eventually synchronous, i.e., during intervals in which messages reach their correct destinations within some fixed (but potentially unknown) worst-case delay.

B. System Architecture

1) Overview: As mentioned before, we integrate OpenLDAP with four BFT protocols: PBFT, Chain, Zyzzyva and Quorum. The integration is similar in the four cases since all our BFT libraries have a common interface that provides two basic methods: *issue_request()* that is called by clients, and *execute_request()* that is used by replicas to execute application-based requests (e.g., LDAP *Read/Update* operations). The system was implemented in *C/C++* code.

2) System Components: *BFT-LDAP* is composed of five components (Fig. 3):

- Client application: a simple demo to construct random LDAP requests, and then calls the BFT client proxy.

- BFT client proxy: receives requests from the client application and delivers them to the BFT server proxies according to the BFT protocol message pattern (Fig. 1).

- BFT server proxy: receives requests from the BFT client proxy and invokes application-based operations via the *LDAP Access Unit*.

- LDAP Access Unit: a mediator that receives the invoked requests from BFT server proxy, and calls the OpenLDAP API to execute the LDAP operations locally.

- OpenLDAP: the application server (also called *slapd server*), executes the operations and returns them back to LDAP Access Unit.

3) Detailed Design: In the following text, we explain the system behavior by describing the path of a request through the different components, starting from the client application to the OpenLDAP *slapd* server, and then its way back.

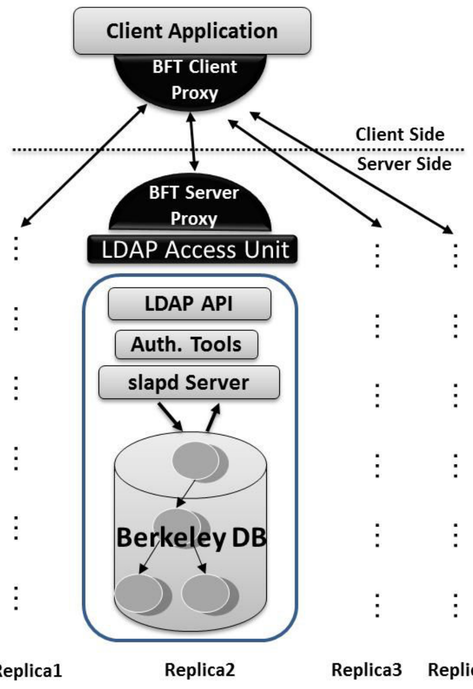


Figure 3. System architecture of BFT-LDAP for f=1.

- First, requests are issued by the client application c. The client that we use is too basic; it represents a function *create_request_command()* that constructs request commands (a number of LDAP entries and sizes).

- The client c forwards the request to the *BFT client proxy* by calling the $issue\_request()$ method through the BFT interface. The proxy encapsulates the command in a special BFT request message after adding the necessary encryption techniques like MAC authenticators and digests (depending on the BFT protocol used), and sends it to the BFT server proxy on the replicas by calling $invoke\_request()$ method, according to the message pattern of the protocol Fig. 1.

- Upon receiving the request form the BFT client proxy, the *BFT server proxy* validates the identity of the messages, and then invokes the $execute\_request()$ function on the *LDAP Access Unit* (i.e., on the local machine).

- The LDAP Access Unit executes *ldap_query()* locally on each replica by binding (i.e., opens a session) to *slapd* server, performing the operation, and then unbinding. The LDAP Access Unit is composed of the API functions: *ldap_initialize()*, *ldap_bind_s()*, *ldap_search_s()*, *ldap_add_s()*, and *ldap_unbind_s()*.

- After executing the request, the slapd server returns the result back to the LDAP Access Unit. The latter sends back the reply to the BFT server proxy which, in its turn, forwards it to the BFT client proxy, again, according to the message communication pattern in Fig. 1. When the BFT client proxy validates the received replies from all replicas to be correct and identical, it forwards the response to the client c; otherwise, the request is considered failed.

Distinct protocols operate differently upon failure detection. PBFT can still commit requests even when less than f+1 replicas are faulty. In case the faulty replica is primary, a *view change* is launched. Chain and Quorum abort upon failure after collecting the *abort history* to initialize a new instance (this complies with the abortability approach [18]).

C. Scalability Issue

To support a higher number of clients, we used the *ldapi* protocol to bind connections to the *slapd* server. In fact, using *ldap* usually opens two connections per communication: one for sending, and another for receiving; this reduces the number of maximum connections allowed (i.e., clients) to less than 30000. *ldapi* is a variant of *ldap*, but less secure, that allows the usage of *UNIX sockets* instead of TCP sockets. This design choice overcomes the limitation of open TCP sockets: 65536. Using *ldapi* is safe in our case since accessing the *slapd* server occurs locally through the LDAP Access Unit. The implementation of ldapi is straightforward using the API *ldap_initialize()* interface.

IV. Experimental Settings

A. BFT-LDAP Installation

We conducted our experiments on 25 $64-bit\ Xeon$ machines with $2GB$ of memory, employed on Emulab [15]. The number of replicas is four, and the fault factor f is equal to one. Each replica runs on a separate machine and the clients share 20 machines. All machines are connected using a star topology into one LAN; in addition, replica machines are connected via another dedicated LAN (this is optimal to get better performance for all protocols).

To evaluate BFT-LDAP, we use the *a/b benchmark*[1] with payload sizes: 0/0, 1/1. This is the same benchmark used in [3]. In the case of 0/0, the payload is actually 64 bytes and not zero, this represents the smallest LDAP command we could get. We do not use any LDAP-related benchmarks since they are not much popular.

Our experiments consider only *search* operations. We do not address *Update* operations for two reasons: first, search operations are the most frequent in Directories (more than 99% of total operations). Second, Update operations may lead to replica inconsistency, and thus the BFT recovery phase is forced. This requires a *Transaction* mechanism (i.e., checkpoints) to retain one-copy semantics on all replicas. Unfortunately, Transactions are not yet implemented in OpenLDAP [16] APIs, hence we leave this issue for future work.

[1]In a/b benchmarks, *a*, and *b* correspond to request size, and response size in KB, respectively.

B. OpenLDAP Configuration

OpenLDAP (v.2.4) is installed on all replicas with the Oracle Berkeley Database (BDB v.4). The databases are identical on all replicas; and they are initialized with $100,000$ entries of 600 Bytes each. OpenLDAP and BDB are configured to provide a maximum performance. Thus, we allow any number of clients, any message size, indexing, and large cashing size (i.e., we use an in-memory database to avoid swapping).

V. Evaluation and Observations

A. Integration Cost

1) Integration Without APIs: Indeed, BFT protocols are complex in nature [18]; any attempt to re-factor them, through embedding in OpenLDAP (that has plenty of features), would definitely add further complications. This contradicts with LDAP design objectives, that address simplicity and light-weigh. Moreover, testing such integrations are quite difficult, and bugs are hard to track. Whereas, it is easier to test each application module (BFT and OpenLDAP) aside when APIs are used. Note that, integrating an already installed service imposes additional complexity, and the service providers might be reluctant to compromise their system stability by taking these integration risks. For all these reasons, we moved towards integrating BFT-LDAP using APIs.

2) Integration Using APIs: BFT-LDAP integration can be viewed as a connection of two black boxes: BFT and OpenLDAP. OpenLDAP encloses the business of the Directory; it is made available to third party applications through a well defined API. On the other hand, the BFT library encapsulates the (complex) fault tolerance techniques, and connects to the outer world through interfaces.

As shown in Section III, our code updates are not significant; they represent the black-colored parts in Fig. 3. On the OpenLDAP side, no changes were needed; thus, OpenLDAP legacy code remained intact. However, some updates were needed on the BFT side: we implemented the *LDAP Access Unit* as a mediator to call the OpenLDAP API, and we updated the BFT client and server proxies, accordingly. We coded our updates using *C/C++*. In particular, 120 lines are added/updated to the BFT common library (80 lines in BFT client proxy, and 40 lines in the BFT server proxy). In addition, the LDAP Access Unit required only 245 lines, supporting the basic OpenLDAP operations: search, modify, add, and delete. To sum up, the needed updates to accomplish the BFT-LDAP integration required, only, 365 lines of $C/C++$ code. The integration required a very modest engineering effort. It took less than one month, and was executed by one engineer (this includes understanding OpenLDAP business, design, implementation, and testing).

Therefore, the integration using APIs is easy and fast. It is convenient for pre-installed services since no need to update

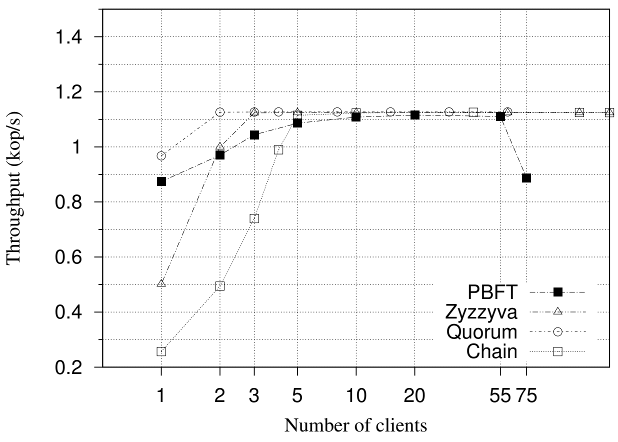


Figure 4. Throughput of BFT-LDAP using the 1/1 benchmark when various BFT protocols are applied (log. scale on x).

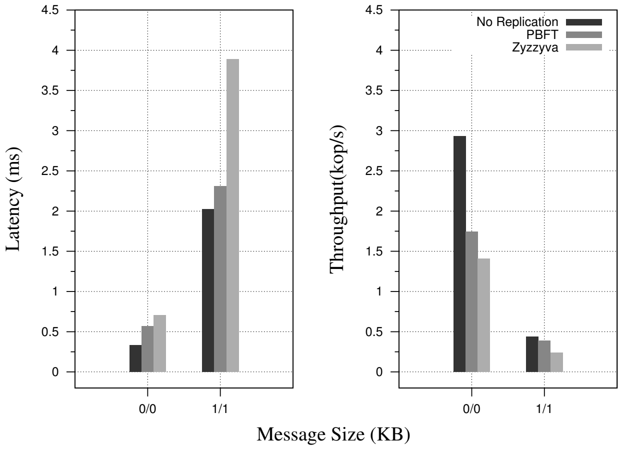


Figure 5. Throughput and latency of BFT-LDAP with 0/0 and 1/1 micro-benchmarks (requests are issued by one client).

the service legacy code while plugging the new updates (i.e., BFT library in our case). In addition, testing the system can be easy; in case the expected results are found to be unsatisfactory, the integrated BFT plug-in can be simply discarded with nil impact on the service.

B. Protocols are Equivalent in BFT-LDAP

We started by studying the behavior of BFT-LDAP with the four BFT protocols: PBFT [3], Zyzzyva [5], Chain, and Quorum [8]. However, our experiments showed that the performance differences, more specifically throughput, are negligible among these protocols. Fig. 1 conveys the throughput of the different protocols (y-axis) as a function of the number of clients (x-axis). With up to 5 clients, Quorum (resp., Chain) achieved the best (resp., worst) throughput among all protocols. However, as more concurrent clients issue requests, the throughput of BFT-LDAP upon using the distinct protocols becomes very close. The difference could not be distinguished in Fig. 1 (even upon trying log. scale on the y-axis too).

These results look astonishing for the first glance since they contradict with previous results [5], [8]. However, a closer look at the system indicates that the high processing time (more than $200\mu s$) of OpenLDAP operations is dominating the performance overhead between the protocols. This overhead is usually caused by the difference in the message patterns of these protocols (Figure 1). We did not get similar results in [5], [8] for instance, since operations of small processing time (less than $20\mu s$) were being used.

We conducted further experiments by changing message sizes. The results looked very close to Fig. 4. Consequently, we confine our study on PBFT and Zyzzyva in the following sections, being the most popular protocols.

C. BFT-LDAP Performance

1) Micro-benchmarks: Fig. 5 presents the 0/0 and 1/1 micro-benchmarks (i.e., with only one client) of the latency and throughput of stand-alone OpenLDAP (i.e., without BFT replication), and with applying PBFT and Zyzzyva. The left histogram shows that the former achieves the lowest latency as compared to PBFT and Zyzzyva, this sounds logical as both protocols impose additional delays due to their message patterns (Subfig. 1(a) and Subfig. 1(b)). The same figure indicates that the latency of PBFT is less than that of Zyzzyva. We refer this to the extra delay in Zyzzyva, since the replicas in Zyzzyva execute the operations sequentially, whereas, in PBFT requests are executed in a single commit phase. This difference appears clearly in our case due to the large processing time of OpenLDAP operations. In addition, Fig. 5 demonstrates that the latencies in the 1/1 benchmark are larger than that of the 0/0 benchmark, this is appears since using larger requests means more complex *search* operations which impose additional processing costs (i.e., $1.5ms$). We mention that the latencies are roughly proportional to the execution time (i.e., $0.2ms$ and $1.5ms$ in 0KB and 1KB messages, respectively).

The right histogram of Fig. 5 conveys the throughput when a single client is sending requests. The opposite logic to the above can be applied here since throughput is the reciprocal of latency in the case of one client [2]; hence, the throughput of stand-alone OpenLDAP is at least double that of PBFT and Zyzzyva (3Kop/s and 0.45Kop/s in the 0/0 and 1/1 benchmarks, respectively). Since throughput is not important in contention-free cases, thus, we do not analyze it further in this case.

2) Maximum Nb. of Clients: Fig. 6 conveys the maximum number of clients that the system can handle and the peak

[2]Clients in BFT do not send a new request until the reply of previous request is received.

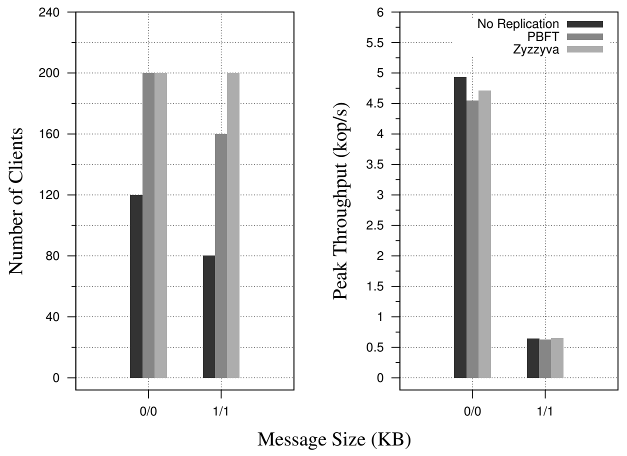


Figure 6. Peak throughput and client scalability of BFT-LDAP with 0/0 and 1/1 benchmarks.

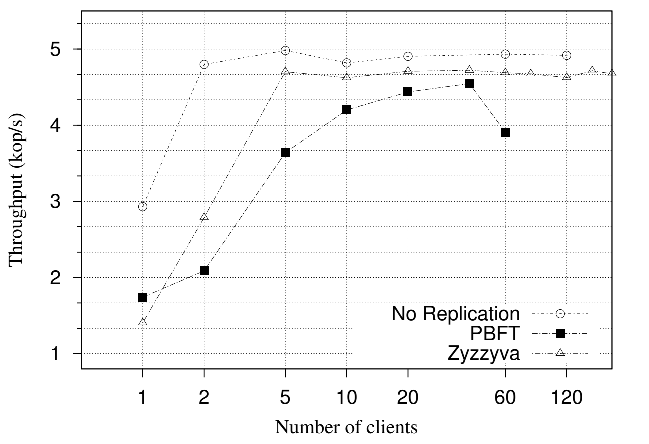


Figure 7. Throughput of BFT-LDAP; 0/0 benchmark (log. scale on x).

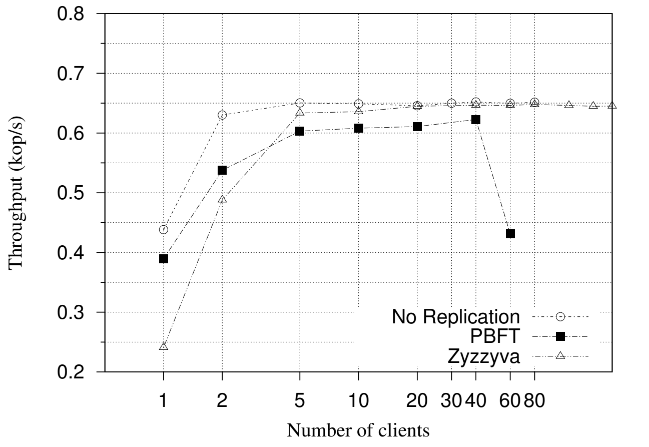


Figure 8. Throughput of BFT-LDAP; 1/1 benchmark (log. scale on x).

throughput using the 0/0 and 1/1 benchmarks. The left side histogram shows that stand-alone OpenLDAP scales up to 120 clients in the 0/0 benchmark, whereas, using PBFT or Zyzzyva increases the tolerance to 200 clients. In fact, OpenLDAP reaches the network bottleneck faster than PBFT and Zyzzyva, since these protocols receive client requests via the primary replica and send the replies back through another replica; while in OpenLDAP stand-alone case, the same NIC is used to receive requests and send replies. Moreover, the same figure shows that using larger messages sizes (e.g., in 1/1 benchmark) increases the traffic on the network, and leads to packet collisions and loss with higher number of clients in stand-alone OpenLDAP or with PBFT cases (80 and 160 clients, respectively). PBFT is affected more than Zyzzyva due to its extensive messaging pattern (SubFig. 1(a)).

3) Peak Throughput: The right hand side histogram of Fig. 6 presents the peak throughput of BFT-LDAP. Identical to Section V-B, the graph shows that the discrepancy of peak throughput (i.e., 5Kop/s) in the three cases is negligible. This is referred again to the high processing time of OpenLDAP operations. Notice that, since larger messages (i.e., complex requests with more *search filters*) requires additional processing time, the peak throughput decreases significantly (almost 6 times) in the case of 1/1 benchmark.

4) Throughput: Fig. 7 and Fig. 8 plot (a logarithmic scale on x-axis) of throughput using the 0/0 and 1/1 benchmarks for stand-alone OpenLDAP or when PBFT or Zyzzyva are applied. Stand-alone OpenLDAP achieves a higher throughput than PBFT and Zyzzyva with up to 10 clients; this is due to the extensive cryptography and the composite messages used in BFT. Also with up to 10 clients, Zyzzyva dominates PBFT because of the complex message pattern of the latter (except for the case of one client which is explained in the previous subsections). With more clients, the throughput curves of BFT-LDAP become very close (diff. less than 10%) whatever BFT protocol is used. This occurs due to the high processing time on replicas that dominates the communication overhead. The behavior in the case of 1KB messges (Fig. 8) is very similar, however, lower throughputs are achieved due to the additional processing time imposed by large messages. We do not have an explanation why PBFT crashes as the number of clients increase (it could be an bug in the code of PBFT [3].

5) Latency: Fig. 9 conveys the 0/0 benchmark curves of the latency of BFT-LDAP (for clarity, a logarithmic scale on x-axis and y-axis is used). With less than 10 concurrent clients, the figure indicates that the response time of OpenLDAP without replication is lower than that when PBFT and Zyzzyva are used; this sounds reasonable since OpenLDAP with BFT replication induces additional message delays due to the long message patterns. As the number of clients increases, the latency becomes closer. In fact, all the curves appear to be coinciding in the graph since

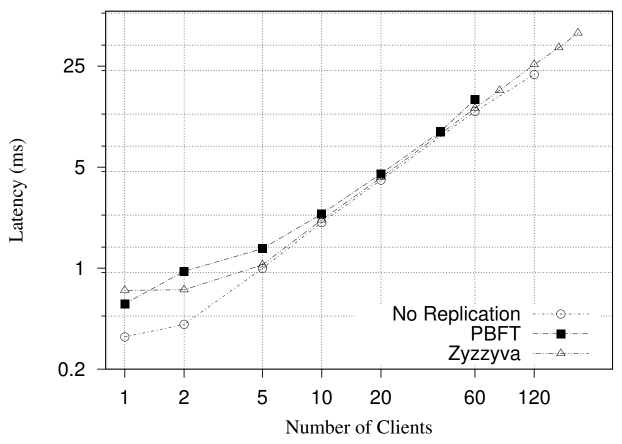


Figure 9. BFT-LDAP response time with 0/0 benchmark (log. scale on x and y).

the processing time of OpenLDAP operations dominates the communication delays with a high number of clients (the requests will be waiting in the OpenLDAP queue). The latencies of PBFT and Zyzzyva increase slightly as the number of client exceeds 40. We explain this by the increasing collisions of packets due to the high number of exchanged messages in PBFT and Zyzzyva as compared to OpenLDAP stand-alone server, this leads to request retransmissions. The results of the 1/1 benchmark are very similar to Fig. 9, thus we exclude it because of the limited size of the paper.

D. BFT-LDAP Replication Cost

Any replication scheme imposes additional costs on the original service. One may claim that applying BFT replication on OpenLDAP (or any Directory Service) is expensive, since it requires more equipment (e.g., at least $3f$+1 replicas to tolerate f faults) and it imposes additional performance overhead. In fact, we believe that quality of service is recommended over cost in critical services. On the other hand, BFT is designed to be deployed on commodity hardware which are very cheap nowadays as compared to the budget of modern (large and intermediate size) enterprises. Cloud computing provides cheap and fast *on-the-demand* [23] solutions too.

Regarding performance, the throughput overhead induced through applying BFT on OpenLDAP is negligible. As demonstrated in Section V-C, the throughput of stand-alone OpenLDAP keeps dominating PBFT and Zyzzyva with few clients. Beyond 10 clients, the throughput of BFT-LDAP with PBFT or Zyzzyva becomes very close to that of stand-alone OpenLDAP. Therefore, OpenLDAP without BFT replication performs only 5% better than Zyzzyva and 10% better than PBFT with small message sizes (see Fig. 7). With larger messages, the throughput curves become even closer (see Fig. 8). The right histogram of Fig. 6 too, conveys that the peak throughput of OpenLDAP is equivalent whether BFT replication is used or not. On the other hand, the left histogram of the same figure shows that BFT replication can boust up the tolernce of OpenLDAP to a higher number of clients; the histogram indicates that stand-alone OpenLDAP crashes with more than 120 concurrent clients, whereas, it scales to 200 clients when Zyzzyva and PBFT are used (in the 0/0 benchmark).

VI. Related Work

Lamport introduced the problem of Byzantine generals in [1]. Liskov et al. [3] developed the first BFT protocol (PBFT) that can handle faults under partial synchrony [2]. The idea was similar to Paxos [24], however with tolerating *Byzantine* faults instead. PBFT requires at least $3f$+1 replicas to tolerate f Byzantine faults ([4], [3]).

Later works like Q/U [6], HQ [7], Zyzzyva [5], Chain [8], and Quorum [8] then appeared to enhance the performance of PBFT by using *speculation*. Upon failure detection, these protocols enter into a recovery phase that operates under failure, but with paying an additional cost. In our paper, we addressed various protocols that represent most of BFT approaches (quorum-based, client-based, primary-based, etc); nevertheless, due to the very comparable performance, achieved between distinct protocols with OpenLDAP, we focused on PBFT and Zyzzyva being the most popular and robust.

The performance of BFT protocols used to be measured using some benchmarks [3], [5], [6], [7], [8], by performing small operations, e.g., incrementing an integer value. Some of them were experimented on NFS system [5], [6]; this gave an intuition about the performance of BFT protocols. This paper, however, tells another part of the truth through studying the behavior of BFT on other application types; e.g., applications with larger processing time like *Directories*.

On the other hand, *Directory* [10] technology proved, recently, a notable growth. Directories can hold critical data that require high reliability. The situation is dangerous since if the confidentiality or availability of the directory service is compromised, then it is likely that the hosting machine is also compromised. This urged us to integrate OpenLDAP (as a Directory) with BFT, seeking high resilience to arbitrary and malicious faults. To the best of our knowledge, this is the first study that addresses Byzantine resilient Directories. An idea about BFT Directories using *COTS* components was introduced in [25], however no practical work appeared afterwards to explore the behavior of this system.

VII. Conclusion

In this paper, we introduced *BFT-LDAP*: an integration of Byzantine Fault Tolerance technology with *OpenLDAP*, the open-source *LDAP-based Directory*. We presented the design of our integration and some technical details about using the OpenLDAP API. Afterwards, we demonstrated that the integration cost is tolerable and that the sacrifice is fairly small. We conveyed that using OpenLDAP API to implement BFT-LDAP was fairly straightforward. In addition, we ensured that the original legacy code of OpenLDAP remained completely intact (this is crucial for services that are already operational). We showed that the difference in OpenLDAP's performance before and after using BFT is negligible. Furthermore, we conveyed the behavior of BFT protocols with Directories; we explored the fact that BFT protocols achieve almost equivalent performance when they are deployed on applications that have high processing time operations like OpenLDAP.

In the future, we are planing to experiment the system in the presence of failures when we implement Transactions, as the OpenLDAP API becomes available.

Acknowledgment

The authors would like to thank Rachid Guerraoui and Nikola Knežević for their valuable discussions.

References

[1] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, 1982.

[2] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.

[3] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[4] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 824–840, 1985.

[5] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 45–58, 2007.

[6] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, "Fault-scalable byzantine fault-tolerant services," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 59–74, 2005.

[7] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, "Hq replication: a hybrid quorum protocol for byzantine fault tolerance," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 177–190.

[8] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *EuroSys '10: Proceedings of the 5th European conference on Computer systems*. New York, NY, USA: ACM, 2010, pp. 363–376.

[9] Ali Shoker and Jean-Paul Bahsoun, "Recover to Self: Re-Abstract Family (regular paper)," in *The International Conference on Computer and Management*, ser. CAMAN. IEEE, March 2012.

[10] International Telecommunication Union, "The directory – overview of concepts, models and services (x.500)," Dec. 1988.

[11] US DoD and DISA, "Directory Services, Security Technical Implementation Guide," 2007.

[12] D. Chadwick, "Threat Modelling for Active Directory," *Conference on Communications and Multimedia Security (CMS 2004)*, pp. 203–212, 2004.

[13] W. R. Claycomb and D. Shin, "Towards secure virtual directories: A risk analysis framework," *Computer Software and Applications Conference, Annual International*, vol. 0, pp. 27–36, 2010.

[14] K. Zeilenga, H. Chu, P. Masarati, and Others, "OpenLDAP," Computer Software, 1998. [Online]. Available: http://www.openldap.org/

[15] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, 2002.

[16] K. D. Zeilenga, "Lightweight directory access protocol (ldap) transactions," RFC Editor, United States, March 2010. [Online]. Available: http://tools.ietf.org/html/rfc5805

[17] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[18] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocol," EPFL, Tech. Rep. LPD-REPORT-2008-008, 2008.

[19] M. Wahl, T. Howes, and S. Kille, "Lightweight directory access protocol (v3)," RFC Editor, United States, 1997.

[20] IBM Corporation, "Understanding ldap - design and implementation," Riverton, NJ, USA, 2004.

[21] OpenLDAP Foundation, "OpenLDAP Software 2.4 Administrator's Guide," 2010. [Online]. Available: http://openldap.org

[22] OpenLDAP Foundation, "Public License for 2.4.23," License, 2010. [Online]. Available: http://openldap.org/software/release/license.html

[23] Amazon.com, "Amazon ec2," 2010. [Online]. Available: http://aws.amazon.com/ec2/

[24] Leslie Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, pp. 133–169, May 1998.

[25] F. Vieira, P. Sousa, and A. N. Bessani, "Transparent byzantine fault-tolerant directory service using cots components," in *Proceedings of the 39th IEEE International Conference on Dependable Systems and Networks (DSN 2009) - Fast Abstract*, Jun. 2009.