BFTCBFTP: BYZANTINE-FAULT-TOLERANT CONSTRUCTION OF BFT PROTOCOLS

EDWARD TREMEL

SIGSEGV 2019

BYZANTINE FAULT TOLERANCE

- Long-standing problem in systems
- Byzantine (adj): excessively complicated, and typically involving a great deal of administrative detail
- Inspired by bickering generals
- Assumes everyone is untrustworthy





BFT PROTOCOLS

- Need to be very complicated
- Much disagreement on how to construct them
- Necessary in order to make faulttolerant systems
 - Because we said so



PROBLEM: CONSTRUCTION OF NEW BFT PROTOCOLS

- Clearly, we always need more BFT protocols
- Constructing a BFT protocol takes a lot of work, hard for one researcher to do alone
- Distributing the work to multiple researchers would help, but systems researchers bicker more than Byzantine generals

Thema: Byzantine-Fault-Tolerant Middleware for Web-Service Applications*

Michael G. Merideth[†], Arun Iyengar[‡], Thomas Mikalsen[‡], Stefan Tai[‡], Isabelle Rouvellou[‡], Priva Narasimhan [†] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA {mom.privalecs.cmu.edu [‡] IBM TJ Watson Research Center, Hawthorne, NY, USA {aruni,tommi,stai,rouvellou}@us.ibm.com

DepSpace: A Byzantine Fault-Tolerant **Coordination Service**

Miguel Correia⁺, Joni da Silva Fraga ¹LaSIGE, University of Lisbon, Lisbon, Portugal ²DAS, Federal University of Santa Catarina, Florianópolis, Brazil

ABSTRACT

The tuple space coordination model is one of the most in-teresting coordination models for open distributed systems due to its space and time decoupling and its synchronization power. Several works have tried to improve the dependability of tuple spaces through the use of replication for fault tolrance and access control for security. However, many practical applications in the Internet require both fault tolerance and security. This paper describes the design and implemen-tation of DEPSPACE, a Byzantine fault-tolerant coordination ervice that provides a tuple space abstraction. The service aervee may provide a type space assuration. The service offered by DEPSACE is secure, reliable and available as long as less than a third of service replicas are faulty. Moreover, the content-addressable confidentiality scheme developed for DEPSACE bridges the gap between Byzantine fault-tolerant replication and confidentiality of replicated data and can be

Categories and Subject Descriptors

C.2.4 [Computer Systems Organization]: Computer

and Communications ! Force Research Labor ing Intrusion Toleranc partially supported the

Keywords

Algorithms, Design, Reliability, Security

Proceedings of the 2005 24th 0-7695-2463-X/05 \$20.00 @

used in other systems that store critical data.

Communication Networks—Distributed Systems; D.4.5 | Software]: Operating Systems—Reliability; D.4.6 [Soft-ware]: Operating Systems—Security and Protection

Byzantine Fault Tolerance, Confidentiality, Tuple Space

Open distributed systems are a fundamental component of our Information Society. These systems are typically com-

posed by an unknown number of processes running in het-erogeneous hosts connected by heterogeneous networks like the Internet. Albeit many distributed applications are still

Permission to make digital or hard copies of all or part of this work for

General Terms

1. INTRODUCTION

Distributed an

Services may call ent parts of the sy

a general strateg tical for the deve

client-server, dist search has been de

of multi-tier, distr

have heterogeneo

the impacts of con

created Thema, a the BFT and Web

tured way to build Services that appl

Services. From a

are also novel in

services: (1) to a

support for their o

SOAP communica

tectural implication

and provide a per

TPC-W benchma

1 Introductio

Distributed app

Services [18, 25]

in different parts

have stringent ava

*Merideth and N

search Office through

Available and Securi

Web Services and

Alysson Neves Bessani¹, Eduardo Pelison Alchieri¹,

programmed using simple primitives like TCP/IP sockets and remote procedure calls, there is an important demand for more powerful tools that allow the design of complex applications with low time-to-market. This requirement more stringent due to the need of guaranteeing tolerance to disconnections, recuperation from server crashes and secu disconnections, recuperation from server crashes and secu-rity against malicious actions. The tuple space coordination model, originally introduced in the LINDA programming language [22], relies on a shared memory object called a tuple space to support coordina-

202

tion between distributed processes. Tuple spaces can sur ort communication that is decoupled in time - pr port communication that is decoupled in time – processes do not have to be active at the same time – and space – processes do not need to know each others locations or ad-dresses [11], providing some level of synchronization at the same time. The tuple space is a kind of storage that stores tuples, i.e., finite sequences of values. The operations sur ported are essentially inserting a tuple in the space, reading a tuple from the space and removing a tuple from the space. The programming model supported by tuple spaces is regarded as simple, expressive and elegant, being supports by middleware platforms like Sun's JAVASPACES [39] and IBM's TSPACES [30].

There has been some research on fault-tolerant [5, 43] and ecure tuple spaces [9, 33, 41], but these works have a narrow focus in two senses: they consider only simple faults (crashes) or simple attacks (invalid access); and they are about either fault tolerance or security. The present pape arous entrer name containes or security. The present paper goes one step further by investigating the implementation of *secure* and fault-tolerant tuple spaces. The solution is in-spired on a current trend in systems dependability that ap-plies fault tolerance concepts and mechanisms in the domain

of security, intrusion tolerance (or Byzantine fault tolerance or security, intrusion continues (or njpamine paironi isternite); [21, 14, 40]. The proposed tuple space is not centralized but implemented by a set of tuple space servers. This set of tuple spaces forms a tuple space that is *dependable*, meaning that it enforces the attributes of reliability, availability, integrity and confidentiality [4], despite the occurrence of Byzantin faults, like attacks and intrusions in some servers. The implementation of a dependable tuple space with th how-mentioned attributes presents some interesting chal-how-mentioned attributes presents some interesting chal-lenges. Our design is based on the *state machine approach*, a classical solution for implementing Byzantine fault-tolerant systems [35]. However, this approach does not guarante the confidentiality of the data stored in the servers: ouit

on the control and y of the data stored in the servers, quite on the contrary, replicating data in several servers is usually considered to reduce the confidentiality since the potentia attacker has more servers where to attempt to read the data

IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 3, NO. 3, JULY-SEPTEMBER 2006

Fast Byzantine Consensus

Jean-Philippe Martin and Lorenzo Alvisi, Senior Member, IEEE

Abstract—We present the first protocol that reaches asynchronous Byzantine consensus in two co Abstract—We prevent the first protocol that reactive asynchronous Byzanine contensus in the communication tapies and tensions. Here, We prevent tour protocol is optimal in time of both number of continuation tapies and number of processes for trea-t contensus. This protocol can be used to build a rejected state in-achine that requires only these communication tapies prevents the tension tapies are made the common case. Further, see show a parameteral variant of the protocol that is add register. (Byzaninis talients and, in the common case, quarantees to-state execution despite some number of failures (±). We show that the guarantee/abstract so-te some case is the optimal optimal of the number of the number of protocols.

ABSTRACT

theoretical minima.

General Terms

Keywords

Performance Reliability

1. INTRODUCTION

Categories and Subject Descriptors

the giving value of data and and shing coeed in hardware make it advantageous for service providers to trade increas-ingly inexpensive hardware for the peace of mind potentially provided by BFT replication. Second, mounting evidence of

Permission to make digital or hard copies of all or part of this work for personal or classroom use in granted without fiel provided that copies are been the stored and the field craiton on the field region. To approximate regulation, to poot on servers or to relistribute to listi, requires prior specific permission and/or field relistions. To show the SOGP '00, October 14-17, 2007, Stevenson, Washington, USA. Occypright 2007 ACM 578-15935-393-304, 2007, Store So. 0.

Zyzzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong Dept. of Computer Sciences University of Texas at Austin {kotla.lorenzo,dahlin,aclement,elwong}@cs.utexas.edu

We present Zyzzyva, a protocol that uses speculation to re 36, 39, 40] suggest that BFT may yield significant benefit even without resorting to n-version programming [4, 15, 33]. Third, improvements to the state of the art in BFT replicaduce the cost and simplify the design of Byzantine fault tolerant state machine replication. In Zyzzyva, replicas retion techniques [3, 9, 10, 18, 33, 41] make BFT replication spond to a client's request without first running an expensive tion techniques [3, 9, 10, 18, 33, 41] make BFT replication increasingly practical by narrowing the gap between BFT replication costs and costs already being paid for non-BFT replication. For example, by default, the Googie file system uses 3-way replication of storage, which is roughly the cost of BFT replication for f = 1 failures with 4 agreement nodes spond to a chem's request without mist running an expensive three-phase commit protocol to reach agreement on the or-der in which the request must be processed. Instead, they optimistically adopt the order proposed by the primary and respond immediately to the client. Replicas can thus be-come temporarily inconsistent with one another, but clients detect inconsistencies, help correct replicas converge on a and 3 execution nodes [41]. This paper presents Zyzzyva¹, a new protocol that uses single total ordering of requests, and only rely on responses This paper presents Zyzzyw^{*}, a new protocol that uses generalized to reduce the cost and simplify the design of BFT state machine replication [19, 35]. Like traditional state ma-chine replication protocols [9, 33, 41], a primary proposes an order on client requests to the other replicas. In Zyzzywa nullie in traditional protocols, may are design with the state cucle requests without running an expensive agreement pro-tocol to definitively entablish the order. As a result, correct that are consistent with this total order. This approach allows Zyzzyva to reduce replication overheads to near their D.4.5 [Operating Systems]: Reliability-Fault-tolerance D.4.7 [Operating Systems]: Organization and Design— Distributed systems; H.3.4 [Information Storage and Re-trieval]: Systems and Software—Distributed systems replicas' states may diverge, and replicas may send different responses to clients. Nonetheless, applications at clients observe the traditional and powerful abstraction of a replicates state machine that executes requests in a linearizable [13]

non-fail-stop behavior in real systems [2, 5, 6, 27, 30, 32

state machine that executes requests in a linearizable [13] order because replies carry with them sufficient history in-formation for clients to determine if the replies and history are stable and guaranteed to be eventually committed. If a speculative reply and history are stable, the client uses the reply. Otherwise, the client waits until the system converges or a stable meth and history

on a stable reply and history. Byzantine fault tolerance, Speculative execution, Replica-tion, Output commit The challenge in Zyzzyva is ensuring that responses to correct clients become stable. Ultimately, replicas are responsible for ensuring that all requests from a correct clien ponsible for ensuring that all requests from a correct client ventually complete, but a client waiting for a reply and history to become stable can speed the process by supplying information that will either cause the request to become sta-ble rapidly which the current view or trigger a view change. Note that because clients do not require requests to commit Three trends make Byzantine Fault Tolerant (BFT) repli-cation increasingly attractive for practical deployment. First, the growing value of data and and falling costs of hardware

but only to become stable, clients act on requests in one o two phases rather than the customary three [9, 33, 41 Essentially, Zyzzyva rethinks the sync [28] for BFT; in stead of pessimistically ensuring that replicas establish a final order on requests before com unicating with a client, w ut commit to the client. Leveraging the client in this way offers significant practical advantages. Comp.

Zyzzyya (ZIZ-uh-yuh) is the last word in the diction According to dictionary.com, a zyzzywa, is "any of various tropi cal American weevils of the genus Zyzzywa, often destructive t

SETUP

- 3f + I systems researchers
 - Why? Because that's the standard for BFT
- Mutually distrustful
- Must agree on details of protocol
- No "trusted third party"
- Solution can't have a leader everyone wants to be the leader
- Everyone has their own public/private key



PREREQUISITE: KEY EXCHANGE

- All BFT protocols depend on signing messages
- Bootstrapping problem: exchanging public keys when the network is untrusted
- Our solution: Researchers meet IRL at a systems conference, give each other keys
- Body doubles impersonating researchers is out-of-scope for this work



PREREQUISITE: KEY EXCHANGE

- All BFT protocols depend on signing messages
- Bootstrapping problem: exchanging public keys when the network is untrusted
- Our solution: Researchers meet IRL at a systems conference, give each other keys
- Body doubles impersonating researchers is out-of-scope for this work



STEP I: BROADCAST STEP I OF PROTOCOL

- Someone broadcasts their proposal for Step I of protocol, signed with their private key
- Whoever takes initiative gets to start



STEP 2: CRITICIZE STEP I OF PROTOCOL

- Each other researcher reads Step I, writes criticism
- Append signed criticism to Step I, broadcast to other researchers
- If anyone receives criticism with different Step I, proof that author of Step I equivocated



STEP 3: HANDLE CRITICISM

- Author of Step I, upon accumulating signed criticism from others, may revise step I in response
- If criticism is contradictory, may choose to reject
- If any criticism agrees, may begrudgingly accept and apply to protocol
- Sends out *revised* Step 1, with signed criticism appended, to prove authenticity of criticism
- Critics may detect equivocation by other researchers on their criticism at this point



STEP 4: REBROADCAST REVISED STEP I

 Other researchers echo the revised
Step I to each other, to ensure author is not equivocating



STEP 5: BROADCAST STEP 2 OF PROTOCOL

- Everyone who has an idea for Step 2 broadcasts it, appended to revised Step I, signed with their key
- Now we have to agree on whose idea to use



STEP 6: VOTE ON STEP 2

- Researchers sign and rebroadcast a version of Step 2 if they agree to use it
- Once a version of Step 2 has signatures from a majority, continue with it
- Decide whether to vote for a version based on reputation system
 - Vote for proposal if you like the researcher who proposed it



STEP 7: CRITICIZE STEP 2 OF PROTOCOL

- Just like criticism on Step I
- Criticize version of Step 2 with majority votes



...AND SO ON





EVERYBODY SENDS LOTS OF MESSAGES TO EVERYONE





HOPEFULLY THIS CONVERGES EVENTUALLY

- Everyone will get the same set of proposals, votes, criticism, etc.
- If enough researchers agree on each step, you can make progress
- But there's no guarantee they will agree
- Oh well, BFT protocols aren't live anyway



EVALUATION

- Somehow, this usually works in practice
- Many papers on BFT algorithms have been written collaboratively



Number of BFT Protocols



I HOPE YOU DON'T HAVE ANY QUESTIONS