Project Vision

for

Team Decided - Raft Consensus Library

1. Introduction

In today's modern world computing is becoming more fundamental to everything we do, and with that comes a focus on creating highly available and distributed services. With high availability services, many computers work together to make a service achieve the greatest uptime possible, this includes being fault-tolerant to various outage scenarios. Consensus algorithms are a foundational part of building these systems.

Consensus algorithm work by ensuring all nodes in a cluster are in agreeance on the current state of the service, and continues uptime of the service during node failure or network link failures between the nodes.

The vision for this project is to create and publish a fully featured implementation of a Consensus Algorithm as an open source code library which would allow for application developers to implement consensus/fault-tolerance into their services as easily as possible. The benefit to an implementing company could be twofold; firstly solving the problem of increasing uptime of a critical service, and secondly by taking advantage of consensus distributed consistent log feature they can maintain service uptime whilst offloading server computation to the end user devices, directly saving hosting costs for the company.

Create a fully featured open source code library to allow the most amount of developers access to adding consensus features as easily as possible to their projects

Joshua and Sean 31-03-2018

2. Positioning

2.1 Problem Statement

Consumer driven requirements for always available services has pushed companies to eliminate any downtime from their offerings. These downtimes impact user experience as well as trust in their platform.

When consumers have poor experiences and lose trust in a service they may consider alternatives solutions for the needs. This impacts the growth of a company, as well as their ability to extract the financial benefits of scale from having a large user base.

Companies which minimise or eliminate any impact from downtime for their end users may better compete in the marketplace for their services.

2.2 Product Position Statement

Our proposed project targets developers who are responsible for implementing services which require high uptime. This open source library aims to fill the market position of a consensus algorithm which focuses on ease of integration and which is written in the language the developer is most likely to be using.

The benefits to a company would include reducing overall product development time, reducing the total cost of developing a highly available service, and by reducing the need/costs for consensus specialists to implement the desired feature. Reducing these costs will also lower the barrier for entry for companies which previously could not have reasonably implemented this.

Driving down development overhead costs can not only improve a company's competitiveness in their market and allow them to capture the cost of unnecessary expenditure, the reduction of development time can also reduce the financial risk that companies accept during their projects which implement this technical feature.

3. Stakeholder Descriptions

3.1 Stakeholder Summary

Name	Description	Responsibility
Project Team	Our team; functioning as developers, as well as project sponsors and management	● Ensuring system is maintainable ● Ensuring market demand ● Ensuring features are working as desired ● Ensuring system is reasonably secure ● Monitoring of project development progress Investing their time into development
Team Leader/CTO	The individual who is responsible for selecting and assessing technologies that may be beneficial for inclusion into their services	● Assess the amount of time and cost of implementation of this library as their feature ● Confirmation of the benefits of the library matching the business requirements ● Comparison of alternative solutions for their requirements ● Proposing library as solution for their requirements to project sponsor ● Managing the project of implementation
Project Sponsor	The individual who has requested the inclusion of the feature into their service, and provide the financing for implementation	● Assess the proposal by the Team Leader/CTO in the context of cost to benefit to the company and realization of benefits ● Responsible to the business for the success of the project and reasonable use of expenditure ● Responsible for sponsoring the resolution of the business case
Developer - Consensus Specialist	The individual in the Project Team whose responsibility is to learn	● Responsible for the intimate knowledge of the library ● Responsible for technical practical knowledge to understand consensus algorithm and implementation ● Liaising as the specialist consult inside the Project Team for practical implementation of the library into the service
Developer	The individuals responsible for the integration of the library into their service	● Understanding their existing service such that they can liaise with the Consensus Specialist to plan how to implement ● Implementing the feature into their service
End user	The end users who is pressuring the company for highly available service	● Not applicable

3.2 User Environment

As the reliance on always available services grows, there becomes more companies and individuals who rely on their services having high uptime. These companies have many, sometimes thousands of users relying on the availability of their services to perform important roles in their lives.

Some examples of the users would be:

- Companies who offload server session to customers, such as multiplayer games.
- Companies offer As A Service (aaS) products needing high uptime
- Companies who want to add high reliability to their services
- Database programmers

Example Scenario 1: The On-Prem Password Manager company

This company produces a Linux webserver that their business clients run on premises, this server is a Password Manager for all of their employees company logins details and supplier logins in. Their biggest customer, a 1000 seat enterprise, has complained of the risk of reliability in this password web server and would like to mitigate the risk of failure of the server as it's stop hundreds of people from conducting their daily duties. They suggested that they were interested in a more reliable solution which mirrors itself between sites to maintain it's uptime, is transaction safe, and importantly never loses data or returns incorrect results.

Example Scenario 2: The database company

Consensus algorithms are complicated, and databases are a critical part of almost all web services. In order to bring consensus to the masses, a database company is looking to implement clustering at the transaction log level of their database, so that web servers can failover to other nodes in the cluster if one falls over. The job of developers greatly simplified as they only need to write heartbeat checks to their database, and upon failure select another node in the cluster.

Example Scenario 3: The Video Game company

There is a video game company who runs a popular multiplayer turn based strategy mobile game, many thousands of players are playing at any one time. Analysing their company costs, hosting costs accounts for around 40% of yearly expenditure, which is second only to staff costs. Each game server hosts up to 10 players at a time, meaning the company needs scale up to run hundreds and potentially thousands of these servers during peak playing times. The idea came up about offloading running the server onto one of the players playing in the server, however it was shot down due to the ability for players to leave at any moment causing bad experiences for the other players in the servers. They've now had the bright idea to employ a consensus algorithm which synchronises server state among their players, so if someone leaves the game server starts running on another player. This will dramatically cut hosting costs, while not impacting uptime for the players.

4. Product Overview

4.1 Needs and Features (Functional requirements)

Priority scale is between 1 and 12, with 1 being the most important

Need	Priority	Features	Planned Release
Consensus between distributed systems	1	• Replicated log, with consensus algorithm	Proto.
Fault tolerant distributed service	2	• Consensus algorithm allows for a fault tolerant distributed system	Proto.
Improved reliability of existing service	3	• System is fault tolerance, so it will improve reliability	Proto.
Complete proven reliability	4	• Based on proven algorithm	Proto.
Minimal additional surface area for failure	5	• Complete coverage unit testing	Version 1.0
Cross Platform	6	• Targeting .NET standard framework	Proto.
Mitigate project abandonment	7	• Licensing allow for profit	Proto.
Minimal overhead/impact to service performance	8	• Equivalent to leading consensus algorithm, Paxos in performance	Proto.
Minimal resource usage	9	• Consensus Log compaction	Proto.
Ability to pick ideal leader	10	• Fitness considered in selection of next leader (new leader calls for election of ideal leader)	Version 1.0
Warm nodes	11	• Tracks log but doesn't vote	Version 1.0
Upgrade path	12	• Versioning built in, backwards compatibility minor releases and single major	Final

4.2 Other Product Requirements (Non-functional requirements)

Priority scale is between 1 and 14, with 1 being the most important

Requirements	Solution	Priority	Planned Release
Reliability	1. Full coverage unit testing	**1**	1. Final
Usability	1. Designed to be as simple as possible to integrate. 2. Released as Nuget package	**2**	1. Proto. 2. Final
Documentation	1. Full coverage documentation for algorithm and API	**3**	1. Final
Quality	1. Full coverage unit testing 2. Strict adherence to style guide	**4**	1. Final 2. Proto.
Performance	1. Matches Paxos in performance of consensus 2. Own thread with ASYNC/non-blocking operations 3. Performance analysis	**5**	1. Proto. 2. Proto. 3. Version 1.0
Compatibility	1. Written in .NET the second most popular language. 2. Minimal dependencies. 3. Written in .NET standard, cross platform 4. Agnostic networking option 5. Designed to be as simple as possible to port languages	**6**	1. Proto. 2. Proto. 3. Proto. 4. Version 1.0 5. Proto.
Availability	1. Can be run between servers locally or across Internet	**7**	1. Proto.
Scalability	1. Dynamic cluster membership, horizontal scaling	**8**	1. Version 1.0
Security	1. Network level authentication	**9**	1. Proto.
Privacy	1. Security measures to join cluster	**10**	1. Proto.
Testability	1. Open source code, unit tests provided	**11**	1. Final
Extendability	1. Open source code	**12**	1. Final
Auditability	1. Open source code 2. Logging	**13**	1. Final 2. Proto.
Troubleshooting	2. Verbose logging	**14**	2. Proto.

4.3 Business Justification for Functional and Non-functional Requirements

First and foremost, the most important feature of this project is increasing the reliability of a User Application Service by maintaining a replicated log; hence the top 5 functional requirements link directly to this. Not only does the desired system require to spread a replicated log amongst nodes, it also needs to be fault tolerant in the case of failure of a minority of nodes,and manage the start/stop of a UAS in cases where only one instance should run. An unconventional part of reliability is usability, therefore there is a focus on the design of the usability of the library to ideally eliminate issues to reliability caused during implementation. To reinforce usability for the product, there will also be complete documentation available.

Companies looking to run this software may look to do so on any operating systems (e.g. Windows, Linux and mobile), so it's important that the code is written in a way that allows for this portability. To implement this, the project will be written in the .NET Standard Framework, which allows portability over all platforms .NET is able to run on.

Part of the benefit of releasing all of the code for this project as open source is that the business risk of project abandonment by ourselves could be picked up by the client, or even a potential open source community. This would enable security updates, bug fixes, and new features to continually be added to the project; and the licensing the project is released under allows for companies to profit from the code.

A fact of implementing a consensus under the hood of a service is that there will always be some overhead; the consensus algorithm being implemented has the same performance as the more popular/complicated and error prone Paxos algorithm. So using Raft gives us all of the benefits, without the understandability/usability downsides of Paxos. This minimal performance overhead allows companies implementing the library to not suffer from unreasonable latency overhead while waiting for servers to reach consensus, which would've caused a direct impact to their user experience. With careful design and the right conditions, it's even be possible to for this library to be used in a real time 30-60 tick game.

When only a single instance of the User Application Server is required, such as a video game server, it may be important to the user experience that most ideal node in terms of latency, or hardware performance runs the UAS. Having the ability for the nodes to pick the most ideal leader for consensus based on performance enable this improved quality of service.

When a node fails or leaves the cluster, this impacts the redundancy/reliability of the cluster as a whole. To bring back a safe state, an additional node is brought in to bring the cluster back up to the desired number of nodes. However, if the cluster service has been running for a long time the replicated log may be so large that it takes several minutes to get another node ready. To resolve this issue, a requirement for allowing Warm nodes to be updated semi-regularly with committed entries to save time later is important. This directly improves reliability and user experience upon additional node failures/departures.

As companies require always on services, an important part is that the consensus library also include backwards compatibility, so allow for rolling software updates through the nodes in a cluster. This allows for the updating of UAS without causing service downtime to the service, avoiding any customer impact.

Security and privacy are also both highly important since these nodes will connect across the network, in a modern day service there is an expectation of a duty of care to users for the protection of their potentially sensitive information or data.

Part of the easy of code maintainability is facilitating the recreation of any possible bugs in the library; as such, to enable the troubleshooting of this the library will output at multiple debugging levels based on desired configuration.

Implementing scalability in the cluster by enabling the dynamic adding and removing of nodes allows the integration of this library with the horizontal scalability necessary for modern infrastructure design.