[Draft Proposal] Exponential Backoff and Jitter for Agent Reconnection

Google Summer of Code Program 2023 Project Proposal

Vandit Singh
vanditsinghkv@gmail.com
Vandit1604

Table of Content

Project Abstract

This project aims to implement Exponential Backoff and Jitter for Agent Reconnection in Remoting Component of Jenkins to reduce the load reconnection attempts when a large controller restarts. It also aims to add Jitter in Swarm Plugin's existing Exponential Backoff Strategy to reduce the chances of collision.

Project Description

A large Jenkins controller is a controller with a large number of agents connected to it. When the controller is restarted for any reason, The connection between All the agents and controller break off. These agents try to reconnect with the Controller at once causing the Thundering Herd Problem.

These reconnection attempts are not effectively handled by remoting as is currently done as a large amount of requests causes collision and increases load on the controller causing controller failures.
This leads to transient failures and connection between Jenkins controller and Jenkins agent cannot be established as a result.

We can solve the Thundering Herd problem with the *Exponential Backoff Strategy*.

Exponential Backoff

Exponential Backoff is an algorithm used in computer networks but in our case it would be used to increase the wait time between the reconnection attempts by agents. In simple terms, If an agent attempts to reconnect to a controller and the reconnection attempts fails. The agent will wait for a calculated amount of time and this time will be calculated by a predefined formula for Exponential Backoff which is:

$$\textbf{Backoff delay} = \textbf{Interval} * \textbf{Exponent}^{\textbf{Retry-attempt}}$$

Exponential Backoff would enable us to avoid collisions between agents' reconnection attempts and would decrease the network congestion and load on the controller side.

Exponential Backoff is nice and dependable. It also increases the efficiency of agent reconnection but still there is one corner case to consider which is when all the agents fail to connect to the controller. Then all of them will try again simultaneously and that won’t be desirable even with the spaced intervals calculated exponentially but if we could add or subtract a calculated random value with each agent’s wait time/delay, synchronization will never happen then we would be able to break the simultaneous retry attempts. This random value is called Jitter.

Jitter

A Random value can be added or subtracted to Backoff Delay to break the synchronization

$$\textbf{Backoff delay with jitter} = \textbf{Backoff Delay} \pm \textbf{Random no.}$$

Exponential Backoff would enable us to avoid collisions between agents' reconnection attempts and decrease the network congestion and load on the controller side.

The swarm plugin already uses the Backoff Strategy to deal with the Thundering herd problem, what it lacks right now is Jitter i.e., randomness in reconnection attempts, there is currently no randomization between the reconnection attempts which only reduces the number of Agents trying to reconnect at a time. Adding randomness between the reconnection attempts of all the agents will increase the chances of connecting to a Jenkins Controller without inducing load on it.

Including this feature in Jenkins can alleviate the load on the controller and minimize conflicts among attempts. With this capability, Jenkins users can proactively prevent transient failures. Currently, when agents fail to connect, users tend to report it as a bug. However, there is a high likelihood that the issue will not persist if they restart the process later.

I chose this project because I find networking really intriguing and learning networking while working on something including networking would be a win-win.

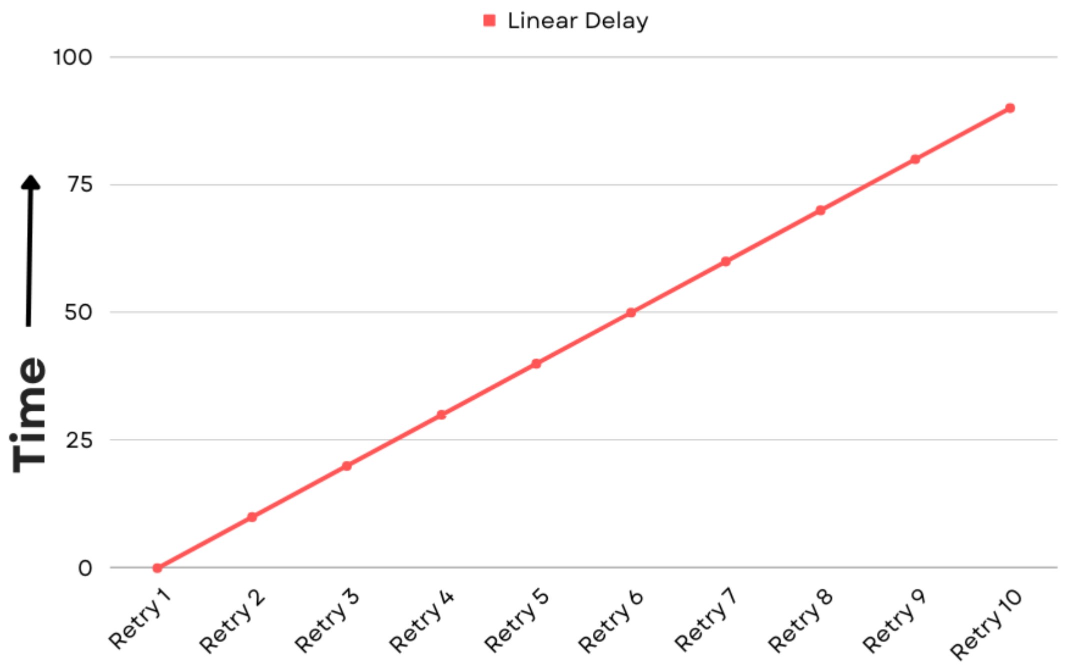
Missing Features in Jenkins

- Inefficient reconnection handling in remoting
- Chances of collision are still high even after the Backoff Strategy in swarm. Adding jitter will reduce the collision.
- No implementation to deal with Thundering Herd Problem

Graphical Representation of Agent Reconnection

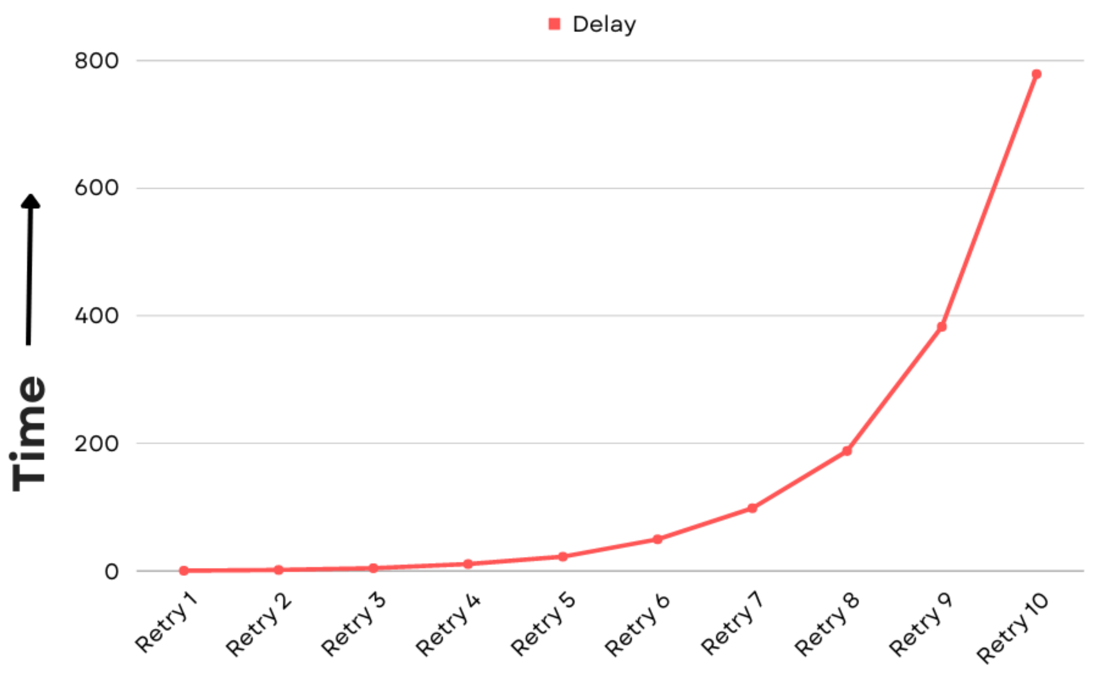
I’ve tried to model the behaviour of the reconnection attempts that happen currently in Jenkins and how my project will improve it.
These values are probable values and not real values. I have calculated them according to the algorithmic trend of xretry techniques in remoting.

1. Linear Agent Reconnection



This Graph shows how things currently work in remoting the agent reconnection is linear it means after every attempt to connect to the controller if connection is rejected, agent waits for 10s and then tries again.

2. Exponential Agent Reconnection



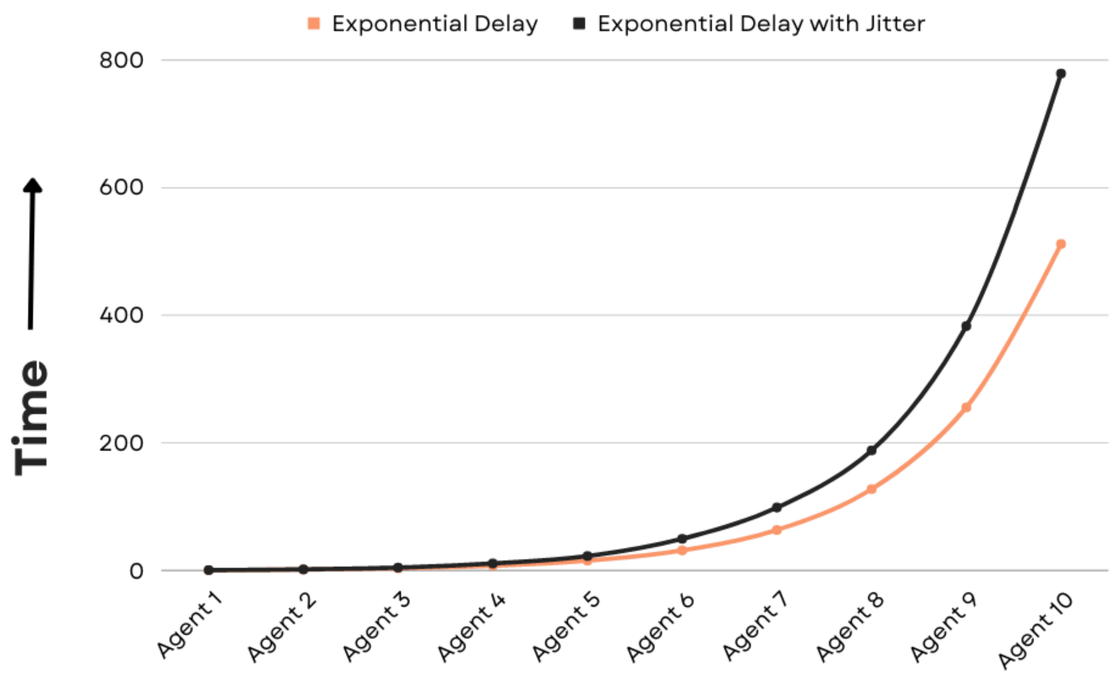
This graph shows how the agent attempting to reconnect will wait for Exponential Intervals. This has two benefits

a. **Collision Avoidance**
b. **Probability to reconnect is very high in initial attempts**

The only corner case we have is for the last attempts the probability to connect is very low however this is only hypothetical for now. Because if no race condition between the agents will happen, the later retries will also connect even with low chances. However if this problem surfaces itself, We can try

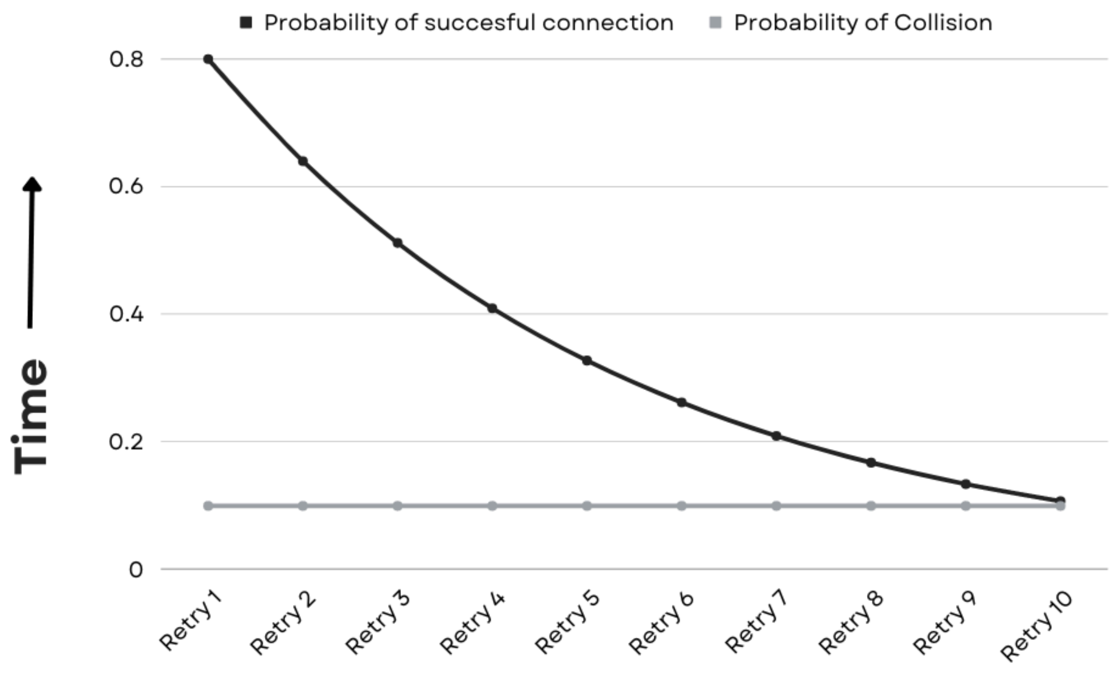
a. **Subtracting the jitter from the delay time**
b. **Ceiling for the delay**
c. **If possible, Chains of backoff would perfectly cater this.**

3. Exponential Backoff v. Exponential Backoff with Jitter



This graph shows the difference between Exponential backoff's performance and Exponential Backoff with Jitter's performance. Backing the increased time if you see in the graph below. You'll see that even if the time increases for later retries the probability of collisions will remain constant at roughly around 0.1.

4. Estimated Probabilities after implementing Exponential Backoff and Jitter



Above graph shows the estimated probabilities after the implementation of Exponential Backoff and Jitter. The Gray line denotes the collisions between agent retry attempts which can be seen as very low. Black line in the above graph shows the probability of successful connection.

Implementation of Exponential and Jitter

Overview

When communicating with external components or services, failures may occur due to various factors such as network issues or server overload. Some of these failures may be considered "hard" errors, indicating a permanent and unrecoverable issue with the service that cannot be resolved through retries.

However, there are also "temporary" failures, known as transient failures, which occur when the external service is temporarily unavailable or fails to connect. These issues are often caused by transient problems such as network congestion or a high number of concurrent requests. While transient failures may cause temporary disruptions in service, they are not indicative of any permanent problem with the service. In fact, retrying the call may result in a successful connection as the issue may have been resolved by the time of the subsequent call.

Retry techniques deal with transient failures very well. Some techniques work better than others, See Tenacity docs explaining all the retry algorithms.

Now, How to deal with Transient Failures? The Solution is Retry Techniques to make sure when you are retrying one of the times the transient failure will cease to exist.
There are lots of retry algorithms now which one to implement?

The Final remoting component I have in my mind is to implement something like the Swarm plugin. Currently if you see in RetryBackOffStrategy.java in Swarm Plugin, it has Three retry logics NONE, LINEAR, EXPONENTIAL but uses the Exponential Backoff as the default retry strategy. Implementing this is essential for unification of retry logic in swarm and remoting.

Phase 1: Stress Testing Framework

What is Stress testing?

Stress testing helps you to evaluate the robustness and reliability of your webserver beyond regular demand. It is usually done to guarantee that the server does not crash while under heavy traffic, as well as to reveal crashes or hangs caused by concurrency difficulties, resource depletion, deadlocks, and other situations.
Apache Jmeter is a great tool for Stress testing on a controller so I'm thinking of using it for Stress testing.

Methodology

1. **Find the endpoint**
   Find the endpoint on which we will load the test and get the time response time which in our case should increase exponentially.
2. **Test the endpoint**
   Using k6 we will have an advantage, as k6 can be used with grafana to have a dashboard to output the stats.
3. **Use grafana dashboard to show the stats**
   The stats can be shown on terminal as well as used with grafana dashboards to show stats.

Todo: use k6 and Jmeter and then choose one of them. Write why I chose the tool. Its benefits and use cases.

Phase 2: Implementing Exponential Backoff in Remoting

As the retrying and connection logic of remoting lives in **Engine.java** after reviewing past PRs attempting to tweak the intervals between retries, I was convinced about that. Some methods which would be changed

- **run()**
  Which startups up the connection between Jenkins Controller and Agents Via Different Set of Defined Protocols and Websocket connection.

- **runWebSocket()**

This method actually establishes a WebSocket connection between a Jenkins agent and the Jenkins controller via **Channel** given by **ChannelBuilder** which is responsible for Communication between Agents and Controllers. Then, the method enters a loop in which it repeatedly attempts to establish a WebSocket connection to the Jenkins controller by creating an **AgentEndpoint** object and using it to connect to the server via a **ClientEndpoint**. The **AgentEndpoint** object is responsible for handling events that occur during the lifetime of the WebSocket connection, such as when the connection is opened, when a message is received, when an error occurs, and when the connection is closed.

If the WebSocket connection is established successfully, a new Channel object is created using a **ChannelBuilder**. The **Channel** object is used to communicate with the Jenkins controller over the WebSocket connection.

If the WebSocket connection is closed, the method attempts to reconnect to the Jenkins controller by restarting the loop.
I'm thinking of rewriting the retrying logic like this:

i) Introducing retries and maxRetries

Introduce retries and maxRetries. As both retries and maxRetries both are self explanatory. Taking 10 retries will be sufficient.

```
int retries = 0;
int maxRetries = 10;
```

ii) We can keep trying to connect to the server and if the connection succeeds we can break out of the loop

```
while (true) {
    try {
        // connect to the server
        hudsonUrl = candidateUrls.get(0);
             wsUrl  =  hudsonUrl.toString().replaceFirst("^http",
"ws");
ContainerProvider.getWebSocketContainer().connectToServer(new
AgentEndpoint(),
ClientEndpointConfig.Builder.create().configurator(headerHandler
).build(), URI.create(wsUrl + "wsagents/"));

          // if we made it here, the connection succeeded, so
break out of the loop
         break;
    } catch (IOException e) {
         retries=retries+1;
         if (retries > maxRetries) {
         throw e;
}
```

If an exception is thrown while connecting to the server, we increment the retries counter and calculate the backoff delay using the exponential backoff formula (2 ^ retries) * 1000. We add some jitter to the backoff delay by adding a random number between 0 and 1000 milliseconds.

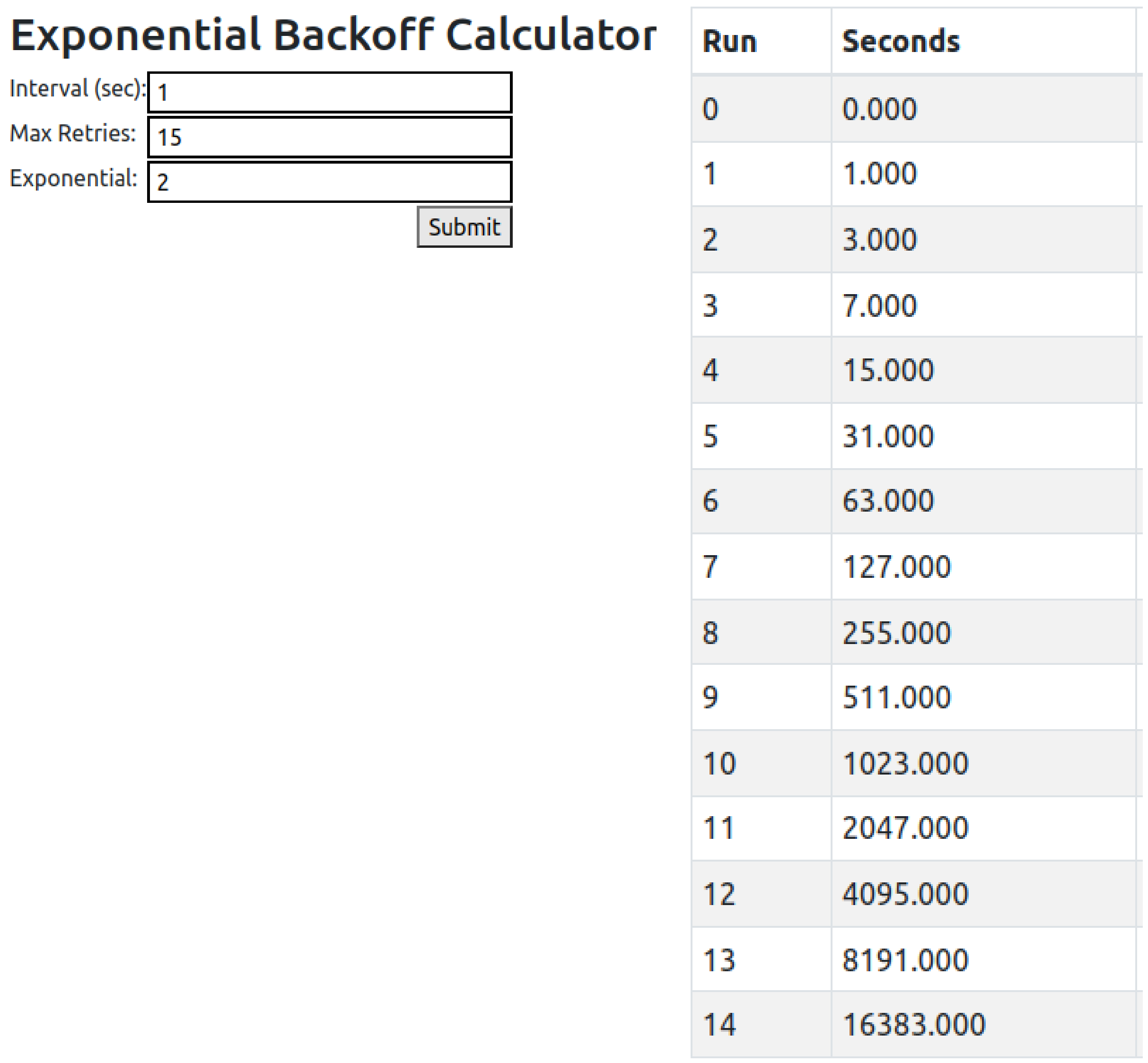
iii) Calculating Backoff delay with Jitter

We can calculate Backoff delay like this, for later retries time intervals can go upto hours so I would use a **min(maxDelay,Backoff_delay)** to use the minimum time so the time interval is capped and would not go till hours of wait time.

$$\textbf{Backoff delay = Interval * Exponent}^{\text{Retry-attempt}}$$

```
// calculate the backoff delay and add jitter
int backoff = (long) (Math.pow(2, retries) * 1000) + (long)
(Math.random() * 1000);
LOGGER.log(Level.WARNING, String.format("Connection to failed,
will retry in %d ms (retry #%d)", backoff, retries), e);
                        try {
                            Thread.sleep(backoff);
                        } catch (InterruptedException ie) {
                            Thread.currentThread().interrupt();
                            throw new
RuntimeException("Interrupted while sleeping during backoff",
ie);
                }
        }
}
```

The problem it can cause if an agent doesn't get connected till 14 retry attempts it will retry again after roughly 4 hours if we consider these parameters of interval starting with 1 second and 2 is the base and the retries goes from 0 to 14. Images are from Exponential Backoff Calculator


Exponential Backoff Calculator

Interval (sec):	1
Max Retries:	15
Exponential:	2

Run	Seconds
0	0.000
1	1.000
2	3.000
3	7.000
4	15.000
5	31.000
6	63.000
7	127.000
8	255.000
9	511.000
10	1023.000
11	2047.000
12	4095.000
13	8191.000
14	16383.000

Considering this maxRetries should be 10 or After a number of retries we can stop using the time interval of the previous attempt but start from 0 until the total retries become 0.
For the reviewer : Should I elaborate more on this ?
[MILESTONE 1]

Phase 3: Unification of Swarm plugin and Remoting

After phase 2, the Process of Unification will be somewhat in the initial stage.

Starting from the swarm plugin which has less complex architecture I can start by

i. Finding Commonalities

As swarm plugin is a remoting wrapper and is built upon the remoting sub-system. It has CLI and Options for choosing the retry algorithm.

ii. Refactoring code

- **Refactoring code in remoting**
  - Deduplication of code in remoting

- **Refactoring code in swarm plugin**
  - The functionalities that are available in remoting and swarm plugins can be backported.

iii. Updating interfaces

We can add the functionalities already present in swarm plugin by extending the existing interfaces in remoting. This will make sure that functionalities present in swarm persist even after the migration to remoting.

iv. Migration

This stage is when the migration is almost at its completion, we would track the bug report to see if migration has caused some issues.

Keeping in mind about the Options for Retrying logic but I would need some guidance in this matter from the mentors so I don't break functionality for some other plugin using swarms and its retrying logics.

i) Unification can be done by changing the methods in swarm to use the methods from remoting if they exist or extending interfaces.

NOTE: To have backward compatibility we won't remove the existing implementation first just add a new one to check if everything is working.

Phase 4: Implementing a chain of Exponential Backoff and Jitter in Remoting and more strategies like Swarm plugin

Implementing Jitter in Swarm Plugin. After the implementation of Exponential Backoff and Jitter, I would like to add the same functionality that swarm plugin has the Option of having 4 Retry logics.

I) NONE

Agent keeps retrying until it connects with the controller with no interval.

II) LINEAR

Agent keeps retrying, taking a time interval between. It keeps retrying until it connects with the controller. The time interval doesn't change with the number of attempts.

III) BACKOFF

Agent keeps retrying after a time interval and the time interval keeps increasing exponentially until it connects with the controller.

IV) CHAIN_BACKOFF

I want to try Introducing a chain of backoffs so an agent doesn't get large intervals between connections. Even though the user can retry via tha Jenkins UI on the controller, Jenkins should be able to deal with this too. I think this will be a great addition to the remoting but I need to research more about this. For reviewer: This is not written in the project idea so should i write this section in Post GSOC section?

I'm looking forward to changing the calculation of Backoff like I have done below by introducing loops. It will shows the behavior like this

Wait 3s for 3 attempts, 7s for the next 2 attempts and 9s for all attempts thereafter

// todo: add code for chain of backoffs

I’m looking forward to implementing the option just like the Swarm plugin using Enums and the OptionHandler approach is suitable for this as implemented in the swarm plugin.

Project Deliverables

i) An updated remoting component with Exponential backoff and jitter implemented.
ii) Unification of swarm plugin and Remoting.
iii) Add more reconnection strategies in remoting. [Stretch goal]

Proposed Timeline

Application Period Begins (March 20 - April 4, 2022):

I have my exams from March 21 - April 1 so I won't be able to contribute much in this time period. However, I have pre-planned my study schedule so that I can set aside 2 hours per day to research more about the project and understand the complex remoting codebase.

Acceptance Waiting Period (April 5 - May 3):

- During this period, I'll discuss more about the project with the mentors and Remoting developers and Swarm developers for better insight into the project and resolve any conflicting ideas.
- I’ll read past discussions related to reconnection of agent to controller.

- I think it is important to learn a bit about the workflow they expect me to follow and the code styles I need to learn.

Community Bonding Period (May 4 - May 28):

- Schedule meetings with the mentors and other community members to get to know them and discuss the project.
- Discuss project approach in further depth with the mentor/s. Figure out if there is any better approach to the problem or the solution could be improved in any way.
- I'll have discussions with long time remoting and swarm plugin developers/maintainers about my approach.
- I'll set up the dev environment for testing and code changes for Coding period 1. I plan to start early making sure I have a buffer of 10 days for my doubts.

Coding Period (May 29 - July 14):

This is draft timeline based on my current knowledge
I would write Tests and Documentation on Weekends and Code on Weekdays

Week	Task	Deliverables
Week 1 & 2 [May 29 - June 10]	☐ Find the endpoint to which agents request for reconnection. ☐ Create benchmarking metric using JMeter or k6(Yet to decide) ☐ Setup grafana dashboard to visualize the information like time taken for reconnection request(full loop POST+GET request)	Benchmarking metric for remoting to demonstrate what happens to the behaviour of remoting with changes to the code. Stress testing framework for stress and load testing, to check the reliability of the remoting and agent reconnection.

Week 3 & 4 [June 11 - June 24]	☐ Implement Exponential Backoff and Jitter in remoting ☐ Implement Jitter in swarm plugin	● Updated remoting component ● Updated swarm plugin
Week 5 & 6 (25 June - 13 July)	☐ Implementing retry logic as they are implemented in swarm plugin using OptionHandler class.	● Updated remoting component with more than 1 retry logic ● Start of unification of swarm plugin and remoting
Week 7 & 8 (14 July - 28 July)	☐ Deduplication of code from remoting ☐ Refactoring code in remoting	● Clean remoting code base so we can start working on extending the existing interfaces

Week 9 & 10 (29 July - 13 August)	☐ Add swarm logics in remoting ☐ Extending interfaces to have swarm functionalities	● Updated Remoting with swarm functionalities
Week 11 & 12 (13 August - 28 August)	☐ Working on unification of remoting and swarm	● Complete Unified remoting and swarm plugin

Final Coding Period for Standard Route (Aug 21 - Aug 28):

I expect the project to be completed successfully by this time. Around this time, I want to finish any remaining formalities from the GSoC or Jenkins sides. The code would already be published on GitHub and available to use.

Future Improvements

- If Chains of Exponential Backoff is not implemented during the program itself i would like to work on it after the program
- Improve and maintain the Stress testing framework.
- Adding CLI to the remoting subsystem.

Continued Involvement

- By contributing to Jenkins I learnt a lot of things and I would love to keep contributing to Jenkins.
- I saw some discussion related to UI/UX so i'll be more than happy to be part of UI/UX sig.
- Also I plan to learn more about DevOps and Contribute to DevOps related repositories in Jenkins.
- I would love to be a part of discussions going on in the community airing opinions and reviewing. Mentoring would be another interesting role in helping contributors. Seeing myself as a maintainer of Jenkins repositories is a great opportunity for me and I would love to adopt a plugin too.

Conflict of Interests or Commitment(s)

My exams will be conducted during mid-August (the final date is not decided yet). During my exams I won't be able to give more than 25 hours for around 2 weeks. Other than that There is no conflicts of interest for me under GSoc 2023.

I commit myself entirely to Jenkins Exponential Backoff and Jitter for agent reconnection for GSOC-2023. I will be available from June to August to work as a committed GSoC intern with 6-7 hours/day. I'll start work early from May. The code I'll write will not be claimed by my university.

Major Challenges Foreseen

- There's a possibility that in future the retry interval for later retries for some agents can go above 60 mins. This behaviour is not
- During unification of retry logics backward compatibility could be a challenge but we can overcome it by thorough research and guidance from the mentors.
- I would like to learn more about the Stress testing framework because right now I don't have much knowledge of it. It will be a interesting challenge through which i can learn something new

---

References

- https://www.jenkins.io/projects/remoting/
- https://www.jenkins.io/doc/book/using/using-agents/
- https://en.wikipedia.org/wiki/Jitter#Testing
- https://www.baeldung.com/resilience4j-backoff-jitter
- https://github.com/jenkinsci/remoting/pull/379
- https://en.wikipedia.org/wiki/Thundering_herd_problem

Relevant Background Experience

I have been contributing to Jenkins for roughly around 5 months. I have good knowledge of Git and Github and I follow all the good practices.
My Journey with Jenkins started in August 2022 and I have learnt so much in these past few months while contributing and exploring various repositories in jenkinsci and jenkins-infra. My Jenkins muscle started with modernizing plugins and later I was solving issues on jenkins.io. I also did some PRs on Jenkinsci, but they were full of loopholes, so I closed them in order to give myself more time to understand the code and improve my overall knowledge of Jenkins and Java.
To sum it all up,I am a problem solver who never gives up. I enjoy learning new things, and I view challenges as opportunities to grow. No matter what comes my way, I am confident that I can overcome it and succeed in the end. I’m a quick learner, and I'm always willing to take on new tasks as opportunities to learn new things. I am certain that I can overcome any challenge and succeed thanks to my perseverance and willingness to learn.

Personal

My name is Vandit Singh and I’m a Junior student of Dr. A.P.J. Abdul Kalam Technical University (India). I’ve been contributing to Opensource for a year. During that one year I learnt a lot of things ranging from Technical Knowledge like good codestyles, understanding code written by someone else to how to interact with people, How to respond to reviews and feedback. I learnt the value of Empathy because Opensource runs on Empathy,Communication and Code Obviously ;)
I learnt about Jenkins when I was exploring DevOps tools. After using Jenkins for my pet project and Interacting with the Jenkins community, I started my contributions to the community and learned a lot of things from many knowledgeble people.

Having the qualities of curiosity, hard work, and persistence can greatly enhance a person's value. As for myself, I possess these three qualities and have demonstrated them effectively while working on issues and pull requests. With these qualities, I am confident that I will be able to successfully complete the GSoC project.

Availability and commitments

My exams will be conducted during mid-August (the final date is not decided yet).
During my exams I won't be able to give more than 25 hours for around 2 weeks. Before that I would be able to devote 30-35 hours a week(4-5 hours on weekdays and 8-9 hours on weekends) throughout the GSoC period and as there would be only one evaluation in between this time, I will try to finish the task before time.

Experience

Open Source Contributions:

- **Merged Pull Requests**

1. jenkins-infra/jenkins.io Fix: Turn off smooth scrolling
2. jenkins-infra/docker-confluence-data [chore] : Redirect Launching Agent from Console

3. jenkins-infra/jenkins.io Clarify top level Pipeline examples of options
4. jenkins-infra/jenkins.io FIX: Add content and links in Building and Debugging Jenkins
5. jenkins-infra/docker-confluence-data chore:redirect building jenkins wiki page to Building and Debugging jenkins.io
6. jenkins-infra/jenkins-io-components feat: Added 'Security' as a tab
7. jenkins-infra/jenkins.io Improve the explanation of section 'agent' in pipeline syntax
8. jenkins-infra/jenkins.io Remove Dead link in sponsors block
9. jenkins-infra/jenkins-io-components fix(footer): Clean up Footer styling
10. jenkins-infra/jenkins.io Pipeline Syntax page - Document that parameters become environment variables
11. jenkins-infra/jenkins.io FIX: Nonexistent scaling of jumbotron images breaks layout
12. jenkins-infra/jenkins.io Patch "Fixed horizontal scroll issue on mobile"
13. jenkins-infra/jenkins.io added typo check and removed htaccess files
14. jenkins-infra/jenkins.io added reference link to docs about agents
15. jenkins-infra/jenkins.io updated the title for better search results
16. jenkins-infra/jenkins.io Correct description of controller/built-in node

- **Open Pull Requests**

1. jenkinsci/embeddable-build-status-plugin add test for PublicBuildStatusAction.java
2. jenkinsci/casdoor-auth-plugin update parent pom to 4.53
3. jenkinsci/conditional-buildstep-plugin update parent pom and spotbugs check
4. jenkinsci/http-request-plugin update parent pom from 4.50 to 4.51
5. jenkinsci/test-results-analyzer-plugin update parent pom from 4.49 to 4.51
6. jenkins-infra/jenkins.io Add information about implied dependencies and detached plugin
7. jenkins-infra/jenkins.io Added Steps for Reporting issue on Github
8. jenkins-infra/jenkins.io Add updating Jenkins section in User's Handbook

Language Skill Set

➢ Java (4/5)
➢ Python(3/5)
➢ Javascript (2/5)
➢ C (3/5)

Tools Skill Set

- ➢ JUnit (3/5)
- ➢ Spring Boot (4/5)
- ➢ Docker (3/5)
- ➢ Git (4/5)
- ➢ HTML & CSS (4/5)
- ➢ Linux (3/5)

Reference Links and Web URLs:

- LinkedIn: vandit-singh
- Instant Messaging: @vandit1604 (Element)
- Timezone: Indian Standard Time (IST), UTC+5:30
- Location: Ghaziabad, India