Distributed Systems

P2: Kubernetes

by
Team 7
Asger Song Høøck Poulsen (202106630)
Firas Harbo Saleh (202109174)

A Distributed Systems
Project

December 6, 2023

Contents

1 Introduction

In the realm of distributed systems, the efficient management and coordination of multiple service instances are paramount for ensuring high availability and resilience. Kubernetes, an advanced container orchestration system, has emerged as a pivotal technology in this domain, enabling scalable and efficient management of containerized applications. Similarly, the Bully algorithm plays a crucial role in distributed systems, providing a robust mechanism for leader election, which is vital for maintaining operational consistency and fault tolerance.

This project delves into the practical application of these concepts through the deployment of a Kubernetes-powered web application - a digital fortune cookie service. By integrating the Bully algorithm within a Kubernetes environment, this project not only demonstrates a novel use case in distributed computing but also explores the synergy between container orchestration and leader election algorithms.

This report will cover the design and implementation of the application, the challenges encountered during the process, and will conclude with a discussion of the results and a brief overview of the lessons learned from the project.

2 Methods and materials

This section of the report elaborates on the comprehensive approach and the array of tools and technologies employed in the development of the Kubernetes-based fortune cookie service, underscored by the Bully algorithm for leader election. The methodology adopted for this project is a testament

to the harmonious blend of software engineering practices, containerization techniques, and orchestration strategies. Here, we dissect the various components and processes that form the backbone of this project, ranging from the Kubernetes infrastructure and the Bully algorithm's integration to the development environment and tools that facilitated the seamless realization of the service.

2.1 Kubernetes

Kubernetes, a cornerstone of modern cloud-native applications, is an open-source container orchestration system for automating software deployment, scaling, and operations of containerized applications. In this project, Kubernetes is not just the infrastructure platform but also an integral part of the application's architecture. The following aspects of Kubernetes play a crucial role in the implementation of the fortune cookie service:

- **Pods**: As the smallest deployable units created and managed by Kubernetes, pods are used to host instances of the application. Each pod runs a containerized version of the fortune cookie service.
- **Replication and Scaling**: Kubernetes manages the desired number of pod replicas, ensuring high availability and load distribution. This is vital for maintaining service continuity, especially during leader election phases.
- **Services and Networking**: A Headless Service in Kubernetes is used for enabling network identity to the pods. This allows pods to discover each other and communicate, which is essential for the Bully algorithm to function.
- **Deployment and Management**: Kubernetes streamlines the deployment process, allowing for consistent and repeatable deployment of the application, and provides tools for monitoring and managing the application's state.

2.2 Bully Algorithm

At the heart of this project lies the implementation of an improved Bully algorithm. The Bully algorithm, described in [1], is a classic method used in distributed systems for leader election. It is characterized by its simplicity, improvement in efficiency, and straightforward approach to determining the leader node in a cluster. In the context of Kubernetes, this algorithm assumes a new dimension of relevance:

- **Election Process**: The algorithm operates by electing the pod with the highest identifier as the leader. When a pod believes it should be the leader (e.g., when the current leader fails), it starts an election process to assert its role.
- **Implementation in Kubernetes**: Implementing the Bully algorithm in Kubernetes involves pods communicating over the cluster network to perform the election process. Each pod must be aware of others and capable of sending and receiving election messages.
- **Handling Failures**: A key feature of this project is the ability of the system to handle leader pod failures gracefully. The Bully algorithm ensures that a new leader is elected swiftly, minimizing downtime and maintaining the availability of the fortune cookie service.
- **Integration with the Service**: The elected leader pod hosts the web interface of the fortune cookie service. This integration demonstrates the practical application of the algorithm in a real-world scenario.

The project's focus on the Bully algorithm showcases its potential in a Kubernetes environment, particularly in handling scenarios involving leader failure and the subsequent re-election process, by ensuring the continuity of the fortune cookie service.

2.3 Web Application

The Web Application serves as the user-facing component of our Kubernetes-based fortune cookie service. It represents the culmination of the project's backend functionalities, providing a simple, interactive and intuitive interface for users to engage with the service. The following aspects of the web application are worth highlighting:

- **User Interface Design**: The design of the web application is focused on simplicity and ease of use. It features a clean layout with a prominent button for requesting a fortune and a display area where the fortune appears.
- **Client-Side Scripting and Interactivity**: JavaScript is utilized to enhance the interactivity of the web application. It handles user events, such as button clicks, and manages the communication with the backend to fetch and display fortune cookies. This scripting ensures that the user interface is dynamic and responsive to user actions.
- **Backend Integration**: The frontend seamlessly integrates with the backend service hosted on Kubernetes. It communicates via HTTP requests, retrieving fortune cookie data from the service managed by the elected leader pod.
- **Frontend Technologies**: The application is built using standard web technologies - HTML for structure, CSS for styling, and JavaScript for functionality. These technologies were chosen for their wide support across browsers and ease of integration.

This web application acts as the gateway for users to interact with the underlying Kubernetes-managed services, illustrating the effectiveness of combining modern web technologies with cloud-native backend systems.

2.4 Development Environment and Tools

The development of the fortune cookie service and the Bully algorithm implementation was done using the following tools and technologies:

- **Docker**: Docker is a containerization platform that allows for the creation and deployment of containerized applications. It was used to containerize the fortune cookie service and the Bully algorithm implementation.
- **Kubernetes**: Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and operations of containerized applications. It was used to deploy and manage the fortune cookie service and the Bully algorithm implementation.
- **Minikube**: Minikube is a tool that enables the running of Kubernetes locally. It was used to create a local Kubernetes cluster for testing and development purposes.
- **Kubectl**: Kubectl is a command-line tool for interacting with Kubernetes clusters. It was used to deploy and manage the fortune cookie service and the Bully algorithm implementation.

- **Git**: Git is a distributed version control system for tracking changes in source code during software development. It was used to manage the source code of the fortune cookie service and the Bully algorithm implementation.
- **GitHub**: GitHub is a web-based hosting service for version control using Git. It was used to host the source code of the fortune cookie service and the Bully algorithm implementation.
- **Visual Studio Code**: Visual Studio Code is a source-code editor. It was used to write the source code of the fortune cookie service and the Bully algorithm implementation.

3 Experiments, results, and discussion

3.1 Kubernetes Configuration

1. **Deployment Configuration**: Defined in the `deployment.yaml` file, the deployment configuration specifies the desired state of the fortune cookie service. It defines the number of replicas, the container image to use, environmental variables, and the container port.
2. **Headless Service Configuration**: Specified in `headless-service.yaml`, this service configuration defines the network identity of the pods. It enables the pods to discover each other and communicate, which is essential for the Bully algorithm to function.

3.2 Bully Algorithm Implementation

The Bully algorithm implementation is encapsulated in the `app.py` file. We were given a template for the implementation in Kubernetes[2] and adapted it to suit the needs of the fortune cookie service. The implementation is based on the pseudocode in figure 1.

The implementation can be found in the `app.py` file.

3.3 Architecture

Overview

The architecture of the fortune cookie service is designed to be scalable and distributed, comprising several components, each serving a distinct role in the service's overall functionality:

- **Frontend**: This is the user interface of the web application, responsible for presenting information and facilitating user interactions.
- **Backend**: The backend service operates within Kubernetes pods. It processes requests, implements the Bully algorithm for leader election, and manages the generation and delivery of fortune cookie texts. The application's state, including the results of leader elections and generated fortunes, is maintained temporarily in the pod's memory during runtime. This means that the data is not persistently stored and will be lost if the pod restarts or stops.
- **Data Storage**: The current implementation of the application does not involve persistent data storage like a database or file system. All operations are performed in-memory, providing fast access but lacking data persistence. However, the architecture allows for the future integration of a database for persistent storage if needed.

```
async function run_bully():
    Perform setup for Kubernetes environment
    Loop indefinitely:
        If the current pod is the leader:
            Serve the website and exit the loop

        Retrieve the list of IP addresses of other pods in the cluster
        For each pod IP in the list:
            Try to get the pod ID from the pod
            If the current leader ID is encountered among other pods:
                Skip to the next iteration of the loop

        If an election is not in progress:
            Change the election status to in progress
            Start the election process
            Select the candidate for the leader

        If the current pod has the highest ID or there are no other pods:
            Serve the website, change election status, and exit the loop

        Wait for a specified duration before the next iteration

    Handle any exceptions that occur during the process
```

Figure 1: Bully Algorithm Implementation

- **Networking**: The application relies on internal Kubernetes networking for communication between pods. This aspect is crucial for the operation of the Bully algorithm, which requires frequent and reliable communication between the pods for leader election and status checks.
- **Docker Hub and Minikube**: Docker Hub is used to store the container image of the application, while Minikube provides a local Kubernetes environment for development and testing. These tools facilitate the deployment and scaling of the application but do not directly influence the in-memory operation of the application.

This architecture ensures a balance between efficiency, scalability, and simplicity, making it well-suited for the dynamic environment of a Kubernetes-based deployment.

Endpoints

The `app.py` file defines the following endpoints for the fortune cookie service:

Note that the '`/`' endpoint is not explicitly defined in the `app.py` file. Instead, it is configured in the `serve_website` function, which is called when the application is run. This function serves the `index.html` file, which is the main webpage of the application on the web-port 8080.

Endpoint	Description
`/pod_id` (GET)	Retrieves the ID of the current pod. Used for pod identification during the election process.
`/receive_answer` (POST)	Receives an answer from a pod during the election process. Helps in determining the current leader.
`/receive_election` (POST)	Endpoint for receiving election messages from other pods. Triggers the election process in the receiving pod.
`/become_coordinator` (POST)	Endpoint for declaring the current pod as the new coordinator (leader). Updates the leader status of the pod.
`/declare_leader` (POST)	Communicates the decision of the election to all other pods. Ensures all pods are aware of the current leader.
`/election_ongoing` (POST)	Signals the status of an ongoing election. Helps in managing the election state across pods.
`/` (GET)	Serves the main webpage (index.html). Primary user interface for the application.
`/receive_coordinator` (POST)	Receives information about the coordinator (leader) pod. Updates the leader information in the pod receiving this request.

Figure 2: Overview of the endpoints in the Bully Algorithm Implementation

3.4 Deployment and Testing Process

This subsection details the critical steps involved in deploying and testing the fortune cookie service within a Kubernetes environment. It highlights the processes of building and deploying the application, followed by load testing and failure simulation, ensuring the system's robustness and reliability.

3.4.1 Building and Pushing Docker Images

The first step in the deployment process is to build the Docker images for the fortune cookie service and the Bully algorithm implementation. The images are then pushed to Docker Hub, a cloud-based repository for storing and sharing container images. This step is streamlined using a `Makefile`.

3.4.2 Apply Kubernetes Configuration

The next step is to apply the Kubernetes configuration files to the cluster. Kubernetes configurations were applied using `kubectl apply` commands for the deployment and the headless service. This step is also streamlined using a `Makefile`.

3.4.3 Load Testing

Load tests were performed by sending rapid HTTP requests to the service endpoint to evaluate the systems' response times and robustness under heavy load.
The test involved the following steps:

1. Increasing the number of threads in orders of magnitude from 10 to 1,000,000 to simulate concurrent users.
2. Sending 1,000 requests per thread to the service hosted on a local Kubernetes cluster.

3. Recording the response time for each request.
4. Calculating the average response time for each thread count.
5. Repeating the test 10 times for each thread count and calculating the average of these averages to ensure consistency.

The results from the load tests were as follows:

Number of Threads	Average Response Time (seconds)
10	0.08
100	0.88
1,000	4.78
10,000	5.17
100,000	4.17
1,000,000	4.99

3.4.4 Failure Simulation

Experiments were conducted to assess the distributed system's resilience by simulating leader pod failures. The process involved identifying the leader pod, forcefully deleting it, and monitoring the system's automated recovery and leader re-election process. This procedure was repeated 20 times to test the consistency of the Bully algorithm's performance.

The average time taken for the system to elect a new leader and recover from the failure was approximately 29.6 seconds, demonstrating the robustness of the system's self-healing capabilities. These findings validate the system's resilience and the effectiveness of the Bully algorithm under failure conditions.

The details of the failure simulation and the system's response are documented on video[3], showcasing the self-healing process in action.

3.5 Discussion

Load Test Results and System Performance Evaluation

The load testing conducted on the fortune cookie service provides a nuanced picture of the system's performance across various levels of concurrency. At lower levels, up to 100 concurrent threads, the service maintains rapid response times, demonstrating its capability to efficiently handle traffic. However, as the concurrency level rises to 1,000 threads and beyond, response times increase, indicating the system is encountering bottlenecks in managing high concurrency levels.

Although there's a modest leveling off of response times at 10,000 threads, suggesting resource saturation or queuing mechanisms at play, a surprising decrease in response times at 100,000 threads points to the system's adaptive optimizations. Yet, this trend reverses once again when faced with one million threads, where response times climb, signaling a threshold where the system's performance begins to degrade due to resource constraints or contention.

From these results, it's clear that strategic scaling is necessary to enhance the system's capacity for handling intense loads. Potential strategies for improving scalability and system efficiency include:

- Reducing the overhead associated with the Bully algorithm during the leader election process, which may involve algorithmic refinements or optimizations in inter-pod communication.

- Implementing horizontal pod autoscaling that can dynamically scale the number of pod replicas based on real-time traffic demands, ensuring that the service can accommodate spikes in user requests without compromising performance.
- Integrating service mesh solutions like Istio[4] to provide advanced traffic management capabilities, including intelligent routing, load balancing, and fine-grained control over inter-service communication.

The intersection of these findings and strategies underscores the importance of a holistic approach to building and maintaining scalable, resilient distributed systems. The project's experiences serve as a valuable reference for future endeavors in cloud-native application development and underscore the dynamic nature of performance engineering in distributed computing environments.

System Robustness and Fault Tolerance

The time taken for leader election, as observed during failure simulations, highlights an area of improvement. While the election process is effective, the duration of nearly half a minute may not be suitable for high-availability systems where even brief downtime can be detrimental.

Future improvements could focus on optimizing the Bully algorithm to reduce this time. Enhancements could include:

- **Algorithmic Optimization:** Refining the Bully algorithm to reduce the number of message exchanges required for leader election, thereby decreasing the overall election time.
- **Network Latency Reduction:** Implementing network policies that prioritize election traffic and reduce latency, ensuring that election messages are delivered and processed faster.
- **Stateful Leader Election:** Introducing stateful mechanisms that remember past leaders and their states, possibly leading to a quicker consensus in re-election scenarios.
- **Proactive Leader Monitoring:** Establishing more aggressive health checks and monitoring to detect leader failure early and initiate the election process sooner.

These potential optimizations aim to not only reduce the time taken to elect a new leader but also to enhance the overall resilience and reliability of the system.

4 Conclusion and perspectives

4.1 Conclusion

This project has successfully demonstrated the integration of the Bully algorithm within a Kubernetes environment to create a resilient, distributed web application. The digital fortune cookie service, powered by this integration, highlights the potential of combining container orchestration with leader election mechanisms to build robust systems. The experiments conducted, including load testing and failure simulation, have provided valuable insights into the system's scalability, performance, and fault tolerance.

Through strategic use of Kubernetes' features such as pods, services, and networking, along with the efficient implementation of the Bully algorithm, the service has shown to be capable of maintaining high availability even under adverse conditions. The load testing has revealed the service's responsiveness at various concurrency levels, pointing to the necessity for further optimization to handle high traffic volumes effectively.

4.2 Future Work

The project has laid a strong foundation for a resilient and scalable distributed system using Kubernetes and the Bully algorithm. However, there are opportunities for enhancement that can further automate the system and minimize manual intervention. Future developments may include:

- **Automation of Service Exposure:** Currently, when the leader pod fails, manual port-forwarding is required to restore access to the service. Future iterations could implement a more dynamic and automated approach to service exposure, such as using Kubernetes Ingress or a service mesh like Istio. These tools could provide automatic rerouting of traffic to the newly elected leader pod without manual intervention.
- **Enhanced Bully Algorithm Efficiency:** Optimizing the leader election process to reduce overhead and election time, ensuring rapid failover and minimal service disruption.
- **Persistent Storage Integration:** Incorporating a persistent storage solution to maintain state and user data across pod restarts and failures, which would be crucial for services requiring data durability.

Incorporating these enhancements will address current limitations and expand the system's capabilities, moving towards a fully autonomous and self-healing architecture that is critical for cloud-native applications operating at scale.

In conclusion, the project stands as a testament to the efficacy of Kubernetes and the Bully algorithm in creating a scalable, fault-tolerant service. The insights gained pave the way for future advancements in cloud-native application development, promising greater resilience and agility in distributed computing environments.

References

[1] P. B. Soundarabai, R. Sahai, T. J, K. R. Venugopal, and L. M. Patnaik, "Improved bully election algorithm for distributed systems," 2014, last accessed 5 October 2023. [Online]. Available: http://de.arxiv.org/abs/1403.3255v1

[2] CML, "Bully algorithm implementation in kubernetes," 2023, [Online; accessed 20 November 2023]. [Online]. Available: https://github.com/thecml/bully-in-kubernetes

[3] Project 2. Accessed on 6 December 2023. [Online]. Available: https://www.youtube.com/watch?v=dvfqK8dEVHY

[4] Istio. Accessed on 6 December 2023. [Online]. Available: https://istio.io/

[5] H. Garcia-Molina, "Elections in a distributed computing system," *Transactions on Computing Systems C-31*, 1982.

[6] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed. Maarten van Steen, 2023, ch. 5.

[7] Kubernetes, "Kubernetes documentation / service," 2023, last accessed 3 December 2023. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/service/#headless-services.