COMP3008J Assignment

22207232 Qiyue Zhu

December 13, 2024

1 Overview

A distributed system offers this recruitment company the opportunity to overcome the limitations of a centralized setup, such as single points of failure, limited scalability, and reduced flexibility. With independent nodes working collaboratively, the system can enhance fault tolerance, improve availability, and enable scalability to meet growing operational demands.
Given the current conditions of the company: **a rural, large-sized department recruitment company with a single office and no IT department**, the solution must prioritize low-bandwidth dependency, ease of implementation, and operational reliability. The report addresses three primary goals:

- **Increased Availability:** Ensuring consistent service delivery despite network interruptions or node failures is critical, especially for a single-office setup where any failure could cascade.
- **Scalability:** Accommodating business growth and ensuring smooth data transactions for a large workforce without extensive re-engineering.
- **Data Security:** Protecting sensitive recruitment data while balancing performance and cost considerations.

This report identifies and compares architectures, protocols, and security strategies to deliver a practical and sustainable distributed system tailored to the company's operational constraints and growth aspirations.

2 Proposed System Design Architecture [1]

Given the rural location and limited IT resources, this company requires a distributed system that balances simplicity, scalability, and resilience. After evaluating multiple architectures, the Client-Server model is recommended for its ease of implementation, with a transition toward Microservices for long-term scalability. Peer-to-Peer (P2P) can serve as a supplementary mechanism for decentralized storage and fault tolerance.1

	Peer-to-Peer (P2P)	**Microservices Architecture**	**Client-Server Architecture**
Features:	- Decentralized, nodes share resources.	- Independent, modular services.	- Centralized, clients rely on servers.
Pros:	- Fault-tolerant, organic scalability.	- Scalable, fault-isolated.	- Easy to manage, cost-effective.
Cons:	- Complex to manage, security concerns.	- Requires expertise, higher overhead.	- Server bottlenecks, limited scalability.
Application:	- P2P's decentralized nature makes it a resilient choice for scenarios with intermittent connectivity, but its complexity may exceed the company's IT resources.	- Microservices are suitable for handling specific tasks like applicant tracking and job postings, allowing modular scalability for growing recruitment needs.	- The simplicity of the Client-Server model makes it ideal for the company's core operations, especially given limited IT resources.

Table 1: Comparison of Scalability Architectures

2.0.1 Evaluation:

- **Scalability:** Microservices outperform the other models due to their modular nature, allowing independent scaling of high-demand components. P2P is resilient but requires significant expertise, while Client-Server may struggle with increased workloads.
- **Availability:** P2P offers decentralized fault tolerance, but Microservices coupled with cloud redundancy ensure higher availability with failover mechanisms. Client-server systems are prone to downtime in case of server failure.
- **Security:** Microservices and Client-Server architectures provide centralized control, enabling robust encryption, role-based access controls, and threat monitoring. P2P, while decentralized, poses challenges in managing security across untrusted peers.
- **Cost Considerations:**
  - P2P scales organically with minimal infrastructure investments but may introduce monitoring costs.
  - Microservices require upfront investment in expertise but offer long-term savings through efficient scaling.
  - Client-Server is initially cost-effective but incurs higher costs with scaling needs.

2.0.2 Recommendation:

For this company, there are 3 main aspects to improve, Scalability, Availability, and Security. A scalable architecture is crucial to handle increasing recruitment data and employee collaboration needs. Availability ensures that the system can serve users with minimal downtime. Security is vital for protecting sensitive client and employee information in a recruitment system.

- **Core Operations:** For the current quick set-up, implement a Client-Server model for its simplicity and ease of management since there is still a lack of IT specialists. But for long-term implementation, gradually adopt a Microservices Architecture for its ability to scale individual components independently and manage workload spikes effectively. Use a Peer-to-Peer (P2P) approach as a supplementary mechanism for decentralized storage or collaboration to enhance fault tolerance. This hybrid solution balances scalability, resilience, and cost-efficiency.

- **Scalability Needs:** Use Microservices for modular and independent scaling of specific functions like applicant tracking, job postings, and data analytics.

- **Availability Enhancements:** Considering:
  - Single Office with Rural Connectivity: Active-passive replication minimizes resource use while ensuring failover capability. Cloud-based redundancy adds resilience for network disruptions.
  - High Employee Collaboration: Active-active replication supports concurrent access but may be overkill for the company's current scale.
  - Cost Sensitivity: Cloud redundancy offers managed failover but may introduce recurring costs that need careful budgeting.

  Use active-passive replication as a primary strategy, augmented by cloud-based redundancy for critical data. This balances cost with operational continuity.

- **Security Measures:**
  Considering:
  - Sensitive Recruitment Data: Encryption ensures client and candidate data are protected both in transit and at rest.
  - No IT Department: Cloud security features, including managed key rotation and monitoring, reduce the burden on non-specialized staff.
  - Compliance and Audit Requirements: Role-based access control helps track and limit data access.
  - Budget Constraints: Invest in cost-effective security solutions suitable for a local setup.

  Employ AES encryption for data at rest and in transit, role-based access controls for internal systems, and cloud-managed security tools for monitoring and compliance.

This hybrid approach addresses the company's current limitations while providing a robust framework for future growth.

2.0.3 Real-World Examples

- Spotify leverages Microservices to independently scale its music recommendation services. Similarly, the proposed hybrid model allows the company to scale specific components while maintaining operational simplicity and security.
- Dropbox employs active-passive fail-over to maintain availability during localized server failures.
- AWS uses built-in encryption and compliance certifications to secure its cloud storage services.

3 Distributed File System

A distributed file system can easily accommodate growing data storage needs without major reconfiguration and has better fault tolerance and more cost-effective storage, so using one is highly recommended. 2

	Hadoop Distributed File System (HDFS)	**Amazon S3**	**Local Storage**
Features:	- Scalable, supports large datasets, fault-tolerant with data replication.	- Fully managed object storage with AWS integration.	- Centralized file storage within the office. - Regular backups to ensure data integrity.
Pros:	- Open-source, cost-effective for large data.	- Pay-as-you-go pricing. - High durability and availability.	- Low-cost and easy to manage. - Direct access reduces latency.
Cons:	- Requires expertise; better for batch processing than real-time access.	- Dependence on a third-party provider. - Costs can escalate with increasing data volumes.	- Limited scalability. - Vulnerable to hardware failures without redundancy.

Table 2: Possible Choices of Distributed File System

3.1 Evaluation:

- Rural location and no existing IT department: In rural areas, the network might be unstable, resources are limited, and IT specialists may not be readily available to educate employees or solve technical problems. Therefore, a robust and easy-to-use file system is more convenient. For this consideration, Local Storage is preferred for its simplicity and low cost, but Amazon S3 is also a strong choice due to its stability, ease of use, and seamless integration with the AWS ecosystem.

- Large-sized department recruitment company with a single office: Considering the large number of employees in a recruitment company, the amount of files can be substantial, and every file should be preserved well. The system should handle large-scale data with great fault tolerance and allow multiple real-time access. For this consideration, Amazon S3 is preferred. Although HDFS is known for its high scalability and great fault tolerance with data replication, it is not as suitable for real-time access, which is more necessary for a large-sized recruitment company. Local Storage may not meet the scalability needs for such a large volume of data.
- Security and cost: Since the files primarily contain sensitive personal information of various clients, adequate protection for that information is essential. Although a large company can afford the price of Amazon S3, a cost-effective method with stricter security, such as HDFS, is preferred. Local Storage can be a good initial option for smaller datasets, but it lacks the advanced security features of cloud solutions.

3.2 Recommendation:

- **Primary Choice:** Amazon S3 is the most suitable option, offering ease of use, scalability, and advanced security features. It supports real-time access and integrates well with other AWS services for seamless expansion.
- **Initial Setup:** Begin with Local Storage for a cost-effective solution to manage smaller datasets. Transition to Amazon S3 as data requirements grow.

3.3 Real-World Example

Netflix: Uses Amazon S3 for distributed storage of media files.

4 Communication Protocols [2]

Efficient communication is the backbone of any distributed system, ensuring seamless interaction among components. In a recruitment-focused system, reliable communication between nodes and services is essential to maintain performance and responsiveness. Below is a comparison of key communication protocols considered for the company's distributed system: 3

4.1 Evaluation:

Considering the company's rural location and limited resources, the selection of communication protocols must balance reliability, simplicity, and performance:

- Rural Network Limitations:
  - RabbitMQ handles intermittent connectivity with its queuing system, ensuring task

	gRPC	**RabbitMQ**	**HTTP/HTTPS (REST APIs)**
Features:	- High-performance, supports real-time streaming. Widely used for microservices.	- Asynchronous message queuing with persistence.	- Stateless, lightweight, synchronous communication.
Pros:	- Low latency, uses HTTP/2, supports multiplexing for high throughput.	- Fault-tolerant, decoupled architecture, ensures reliable message delivery.	- Simple, widely supported, secure with HTTPS.
Cons:	- Requires expertise, less suited for non-streaming tasks.	- Requires broker maintenance, adds overhead, not ideal for real-time.	- Higher latency, limited scalability for frequent communication.

Table 3: Comparison of Communication Protocols

completion despite outages.
- REST APIs, being stateless, can gracefully recover from disruptions through retries.

- Scalability Needs:
  - gRPC excels in scalable, high-performance setups but requires stable, low-latency networks.
  - REST APIs provide adequate scalability for moderate workloads, aligning with the company's current needs.

- Ease of Management:
  - REST APIs require minimal setup and infrastructure, making them suitable for a company with limited IT resources. - RabbitMQ offers reliability for background tasks but demands additional management.

- Real-Time Communication:
  - While gRPC supports real-time interactions, its complexity may not justify its implementation for a single-office setup.
  - REST APIs can handle real-time needs acceptably with lower technical overhead.

- Asynchronous Messaging: RabbitMQ is well-suited for decoupled, background tasks like sending email notifications or processing job applications. Its reliable message queuing ensures delivery even during network disruptions.

4.2 Recommendation:

- **Primary Protocol:** REST APIs, due to their simplicity, widespread compatibility, and ability to function well under variable network conditions. - **Supplementary Use:** RabbitMQ for asynchronous tasks like notifications and data synchronization during outages.

- **Future Expansion:** Evaluate gRPC if real-time, high-performance interactions become critical as the company scales.

4.3 Real-World Example:

Uber exemplifies the use of gRPC for real-time communication among its microservices, ensuring seamless interactions in a high-demand environment. Similarly, LinkedIn utilizes RabbitMQ to process millions of asynchronous messages efficiently.

5 Critical Evaluation

The proposed hybrid architecture, combining Client-Server and Microservices models with REST APIs and RabbitMQ, addresses the company's current needs while providing room for growth. The strengths and weaknesses of this approach are outlined below: 4

Strengths	**Weaknesses**
- Scalability: Microservices allow independent scaling of recruitment-specific functions, while REST APIs handle moderate workloads effectively. - Availability: Cloud-based redundancy and RabbitMQ's messaging ensure system uptime even with intermittent connectivity. - Security: Centralized controls in the Client-Server model enforce robust encryption and access management. - Ease of Adoption: A Client-Server setup is straightforward to manage and suitable for the rural office's limited IT capacity.	- Reliance on cloud services introduces costs, vendor lock-in, and reduced infrastructure control. - Microservices add complexity requiring expertise that the company may lack. - Network reliability may hinder real-time protocols like gRPC in rural areas.

Table 4: Evaluation of the Proposed Architecture

6 Recommendations for Implementation

6.1 Short-Term Actions:

- Implement a local Client-Server setup for core workflows, supplemented by cloud storage for backups.
- Use RabbitMQ for asynchronous tasks like notifications.

- Train staff on basic cloud interfaces and distributed system management.

6.2 Long-Term Strategy:

- Gradually expand Microservices for advanced recruitment features.
- Invest in IT expertise to optimize and maintain the infrastructure.
- Enhance security through Zero Trust Architecture and regular system audits.

7 Conclusion

This hybrid architecture provides a pragmatic solution for the company, balancing immediate needs with future scalability. It ensures operational efficiency while addressing the challenges of a rural setting and limited IT resources.

References

[1] Richman, J. (2023). What Are Distributed Architectures: 4 Types & Key Components. [online] estuary.dev. Available at: `https://estuary.dev/distributed-architecture/`.

[2] geeksforgeeks (2023). Communication Protocols In System Design. [online] GeeksforGeeks. Available at: .