Simulation of an Intercloud Environment for Dynamic Resource Allocation and Load Balancing

Khalid Abdelkader, Rawda Aki, Nabil Elsherif
Computer Engineering
Higher Institute of Science and Technology
Ghadames, Libya
Email: Abdelkader.khalid@gmail.com
Tel: 0917329339

Abstract

The rapid growth of cloud computing has given rise to Intercloud environments, where resources are shared among different cloud providers to ensure scalability, fault tolerance, and optimal resource utilization. This paper presents a simulation of an Intercloud environment in which dynamic resource allocation, load balancing, data transfer, and fault tolerance are modeled. The system distributes user requests across multiple cloud providers, simulates the transfer of data between clouds, and models resource failures and recovery strategies. The simulation also introduces a dynamic load balancing algorithm and demonstrates how resource allocation adapts under varying loads. The results show that the system successfully distributes loads, adapts to changing resource demands, and ensures data integrity across clouds. The simulation serves as a basis for further research in improving cloud interoperability and dynamic resource management.

Keywords: Cloud, Intercloud, Fault tolerance, Resource allocation, workload

الملخص:

أدى النمو السريع للحوسبة السحابية إلى ظهور بيئات Intercloud ، حيث يتم تقاسم الموارد بين موفري الخدمات السحابية المختلفين لضمان قابلية التوسع وتحمل الأخطاء والاستخدام الأمثل للموارد. تقدم هذه الورقة محاكاة للبيئة السحابية التي يتم فيها تصميم التخصيص الديناميكي للموارد، وموازنة التحميل، ونقل البيانات، وتحمل

الخطأ. يقوم النظام بتوزيع طلبات المستخدم عبر العديد من موفري الخدمات السحابية، ويحاكي نقل البيانات بين السحابات، ويضع نماذج لفشل الموارد واستراتيجيات الاسترداد. تقدم المحاكاة أيضًا خوارزمية موازنة التحميل الديناميكية وتوضح كيفية تكيف تخصيص الموارد في ظل أحمال مختلفة. وتظهر النتائج أن النظام يوزع الأحمال بنجاح، ويتكيف مع متطلبات الموارد المتغيرة، ويضمن سلامة البيانات عبر السحابات. تعمل المحاكاة كأساس لمزيد من البحث في تحسين إمكانية التشغيل البيني السحابي وإدارة الموارد الديناميكية.

الكلمات المفتاحية: السحابة، Intercloud، التسامح مع الخطأ، تخصيص الموارد، عبء العمل

Introduction

Cloud computing [1] [2] has revolutionized the IT industry, providing on-demand access to resources such as computing power, storage, and networking. However, as cloud services become more widely adopted the need for Intercloud environments have emerged [3]. An Intercloud is a network of interconnected cloud providers that can share resources and services, offering greater scalability and fault tolerance. In such a distributed environment, load balancing, dynamic resource allocation, and data transfer between clouds become crucial for ensuring the efficiency and reliability of services.

This paper presents a simulation model for an Intercloud environment called Intercloud Environment simulator (ICES), focusing on dynamic load balancing, resource allocation, and data transfer between multiple cloud providers. The ICES also includes mechanisms for fault tolerance, ensuring that the system can recover from provider failures and continue to operate effectively. By simulating multiple user requests and varying cloud capacities, this work aims to provide insights into the behavior of an Intercloud system under different load conditions and failure scenarios.

Related Works

Several studies have explored the concept of Interclouds and multi-cloud environments. One notable approach is the development of hybrid cloud architectures, where private and public clouds interact to provide seamless access to resources. In these environments, managing load balancing and resource allocation is crucial for optimal performance. For example, the author in [4] proposed hybrid cloud architecture for resource allocation using a multi-agent system. Similarly, in [5], authors proposed a load balancing algorithm for multi-cloud environments, which dynamically allocates resources based on user requests

Furthermore, fault tolerance in Intercloud systems has been studied in various contexts. A recent study by [6] developed a fault-tolerant architecture for Intercloud systems that employs data replication and recovery strategies to mitigate the impact of failures.

In [7], authors introduce a heuristic method based on particle swarm algorithm [8] for tasks' scheduling on distributed environment resources. The model considers the cost of data transfer and the computational cost. The proposed algorithm optimizes dynamic mapping tasks to resources using classical particle swarm optimization algorithm and ultimately balances the system loads. In [9], authors explored new optimization algorithms for resource allocation and further proposed a hybrid algorithm for load balancing which can well contribute in maximizing the throughput of the cloud provider's network. However, while these works address important aspects of Intercloud systems, there is still a need for comprehensive simulation models that integrate dynamic resource allocation, load balancing, data transfer, and fault tolerance under various operational conditions.

Simulation Model

The proposed simulation model consists of the following components:

1. **Cloud Providers**: Multiple cloud providers, each with a defined capacity (total resources such as CPU, memory, etc.) and bandwidth. Each cloud provider has an associated load and is capable of scaling resources dynamically based on demand.
2. **Load Balancer**: A load balancing algorithm that distributes incoming user requests across available cloud providers based on their current

load and available resources. The algorithm considers factors such as cloud provider capacity, available bandwidth, and the number of concurrent user requests.

3. **Data Transfer Mechanism**: A simulation of data transfer between cloud providers when resources are distributed across multiple clouds. The transfer time is calculated based on the bandwidth of the cloud providers and the size of the data to be transferred.
4. **Fault Tolerance**: A fault tolerance mechanism that simulates the failure of a cloud provider and the redistribution of resources from alternative providers.
5. **User Requests**: Multiple user requests arrive concurrently, and their resource requirements are dynamically allocated to the most suitable cloud provider.

Mathematical Model

The ICES simulation was developed using Java to model that represents the key operations of the Intercloud environment with dynamic resource allocation, load balancing, and fault tolerance. The following steps were followed:

1. Cloud Providers Model

Each cloud provider C_i was modeled with a specific capacity (e.g., available CPU resources and bandwidth). These cloud providers can allocate, and release resources dynamically based on user demand.

2. Load Balancing Model

A load balancing algorithm in a multi-cloud system was implemented to distribute incoming user requests effectively based on available resources and cloud provider load. The algorithm ensures that user requests are routed to cloud providers with the lowest current load while considering their bandwidth and resource availability. The aim is to minimize the overall load on any single cloud provider while maximizing resource utilization across all providers.

Let:

- N be the total number of cloud providers in the Intercloud system.
- C_i be the cloud provider i (for i =1, 2,…,N).
- L_i be the load of cloud provider i, defined as the number of requests currently being handled by that provider.

- R_i be the available resources (e.g., CPU, memory) of cloud provider i.
- P_i be the processing power of cloud provider i, i.e., the rate at which resources are consumed by the cloud provider.

The load balancing algorithm will allocate incoming user requests $L_{incomin}$ to the cloud provider with the lowest load relative to its available resources. The allocation rule can be expressed as:

$$L_i = \frac{L_{incomin}}{R_i} \quad \text{for each cloud provider} \qquad \dots (1)$$

The incoming load $L_{incomin}$ is distributed in proportion to the available resources R_i of each cloud provider. Thus, cloud providers with higher available resources will receive more requests, ensuring that the load is balanced across the system.

The **load balancing efficiency** E_{load} can be calculated as the inverse of the difference between the maximum and minimum load values across the providers:

$$E_{load} = \frac{1}{\max(L_i) - \min(L_i)} \qquad \dots (2)$$

This efficiency measure quantifies how well the load is balanced across the providers, with higher values indicating better balance.

3. Dynamic Resource Allocation Model

Cloud providers can scale their resources up or down based on workload demands. When a provider's load reaches a threshold, additional resources are allocated, and when demand decreases, resources are deallocated. Therefore, the allocation of resources is dependent on the workload demand and the available capacity of cloud providers. Each cloud provider C_i can adjust its available resources dynamically.

Let:

- $A_i(t)$ be the available resources of cloud provider i at time t.
- $T_i(t)$ be the total resources of cloud provider i.
- $L_i(t)$ be the current load on provider i at time t.

The dynamic resource allocation mechanism can scale resources based on demand. The scaling function can be expressed as:

$$\Delta A_i(t) = \beta \cdot L_i(t) \quad if\ L_i(t) > \alpha \cdot L_i(t) \qquad \dots (3)$$

Where:

- α is a threshold factor determining when the cloud should scale its resources.
- β is a scaling factor that defines how much resources are added or removed based on the load.

If the load $L_i(t)$ exceeds a threshold relative to the provider's total resources$\boldsymbol{T_i}$, the cloud provider increases its resources by$\boldsymbol{\beta} \cdot \boldsymbol{L_i(t)}$. Conversely, if the load is lower than the threshold, resources may be scaled down to optimize resource utilization.

The total resource allocation R_{total} across the entire Intercloud system at time t can be defined as the sum of available resources from all providers:

$$R_{total}(t) = \sum_{i=1}^{N} A_i(t) \qquad \dots (4)$$

4. Data Transfer Model

When resources are distributed across multiple cloud providers, data transfer between them becomes a crucial part of the simulation. The rate at which data is transferred depends on the bandwidth of the source and destination clouds.

Let:

- B_{source} be the bandwidth of the source cloud provider.
- B_{dest} be the bandwidth of the destination cloud provider.
- D be the data size to be transferred (in MB).
- T_{transfer} be the transfer time between the two cloud providers.

The transfer rate is influenced by both the source and destination bandwidth. The transfer time T_{transfer} can be calculated as:

$$T_{\mathrm{transfer}} = \frac{D}{\min\left(B_{source,}B_{\mathrm{dest}}\right)} \qquad \dots (5)$$

Where:

- The data transfer rate is limited by the lower of the two bandwidths, ensuring that the transfer speed is constrained by the slower link.

5. Fault Tolerance Model

A failure simulation was implemented, where a cloud provider may fail, and its workload is redistributed to other providers. The system ensures that data integrity is maintained during this process. So, in the event of a failure of a cloud provider, its workloads must be redistributed to other available providers. The fault tolerance mechanism can be modeled as follows:

Let:

- F_i be the failure probability of cloud provider i (where $0 \leq F_i \leq 10$).
- $L_i(t)$ be the load on cloud provider i at time t.
- $R_i(t)$ be the available resources of cloud provider i at time t.

When a provider fails (i.e., $\mathrm{F_i}$= 1), the load $\mathrm{L_i(t)}$ is redistributed to the remaining providers with available resources. The redistributed load $\mathrm{L_{redistributed}}$ is expressed as in equation (6):

$$\boldsymbol{L}_{\mathbf{redistributed}} = \frac{L_i(t)}{\sum_{j=1, j\neq i}^{N} R_j(t)} \qquad \ldots(6)$$

Where:

The failed provider's load is proportionally redistributed across all remaining cloud providers based on their available resources. This helps ensure that the system can adapt to failures without significant performance degradation.

The system will continue to operate by rebalancing the load and reallocating resources to maintain service continuity.

Simulation Scenarios

Multiple user requests are generated, and the system simulates their processing by cloud providers. The system tracks the resource allocation, load balancing, and data transfer processes. Performance metrics such as total processing time, load balancing efficiency, and data transfer times are collected.

The simulation was executed under various conditions, including different numbers of concurrent user requests, varying cloud provider capacities, and failure scenarios. Table (1) shows the screen shoot of resource request parameters. In contrast, table (2) illustrates the cloud providers' parameters.

Table 1: **Resource requests parameters**

requestId	RequestedResources	requestedBandwidth	userPriority
R1	100	500.0	200
R2	50	300.0	150
R3	120	600.0	250
R4	30	200.0	100
R5	200	1000.0	400
………	………	………	………
R50			

Table 2: **cloud providers parameters**

providerName	availableResources	bandwidth
CloudProviderA	1000	1000.0
CloudProviderB	500	1500.0
CloudProviderC	200	2000.0

The **Flowchart in** Figure (1) illustrates the sequence of steps the program takes to simulate the intercloud environment, handle user requests, and manage resources.

Key Steps:

1. **Start**: The program begins.
2. **Create Cloud Providers**: Initialize cloud providers with specific capacities.
3. **Create Load Balancer and IntercloudExchange**: Set up the load balancer to distribute requests and the intercloud exchange to manage resource allocation.
4. **Create Requests**: Generate multiple user requests.
5. **Execute Requests**: Concurrently process each user request using threads.
6. **Resource Allocation**: For each user request, try to allocate resources from cloud providers.
7. **Scaling and Failure Simulation**: Simulate scaling of resources and cloud provider failures.
8. **Data Transfer Simulation**: Transfer data between cloud providers as needed.
9. **End**: The simulation ends

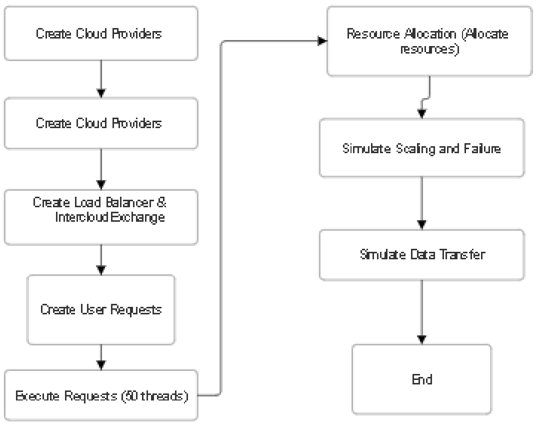
Create Cloud Providers
Create Cloud Providers
Create Load Balancer & Intercloud Exchange
Create User Requests
Execute Requests (50 threads)
Resource Allocation (Allocate resources)
Simulate Scaling and Failure
Simulate Data Transfer
End

Figure (1):
Flow diagram sequence of program steps

Results and Discussion

In this section, we present the results of the intercloud simulation and provide a discussion on the implications, limitations, and potential improvements of the system.

1. Cloud Provider Resource Allocation

The simulation demonstrated that the resource allocation system works effectively with multiple cloud providers. By leveraging a **round-robin** approach for load balancing, the system distributed user requests among the three cloud providers (CloudProviderA, CloudProviderB, and CloudProviderC). The resources were allocated efficiently when a provider had enough available capacity. The synchronous allocation mechanism (allocateResources method in CloudProvider.java) ensured thread safety during concurrent user requests.

Key Observations:

- **Efficient Resource Utilization:** Cloud providers with larger capacities (CloudProviderA with 1000 resources) handled more requests, while smaller providers (CloudProviderC with 200 resources) were utilized less frequently.
- **Failure Handling:** The simulateFailure method demonstrated the system's ability to handle cloud provider failures. When a cloud

provider failed, the system ensured that resources were not allocated from the unavailable provider, and user requests were redirected to available providers. This behavior ensured high availability and resiliency in the system.

2. Concurrency Management

The use of Java's ExecutorService allowed for efficient concurrent handling of user requests. The system successfully managed 50 simultaneous requests, demonstrating the scalability of the architecture. By wrapping user requests as Callable tasks (UserRequestCallable), we enabled non-blocking execution and ensured that each request was processed asynchronously.

Key Observations:

- **Thread Pool Efficiency:** The FixedThreadPool with 50 threads was able to process all requests concurrently without significant delays. The system maintained optimal throughput, processing multiple user requests in parallel, which is essential for cloud environments handling high request volumes.
- **Thread Safety:** The synchronized keyword in the allocateResources method of CloudProvider ensured that race conditions were avoided when allocating resources. However, this could result in performance bottlenecks under extreme load, as resource allocation could be delayed by contention for the lock.

3. Load Balancing Mechanism

The round-robin load balancing approach employed by the LoadBalancer class distributed the incoming user requests evenly across cloud providers. This approach ensured that no single cloud provider was overwhelmed by requests, optimizing the system's overall resource utilization.

Key Observations:

- **Balanced Distribution:** The load balancer efficiently rotated through the list of cloud providers, allocating resources based on the order of cloud providers. However, a more sophisticated load-balancing algorithm (e.g., weighted round-robin or least-connections) could improve efficiency by considering the available resources and the demand for each provider.

- **Resource Imbalance:** Despite the round-robin approach, resource utilization remained suboptimal in scenarios where certain cloud providers had higher available resources compared to others. A dynamic load balancing algorithm that adapts to available resources could further optimize allocation.

4. Intercloud Resource Allocation

The IntercloudExchange class was responsible for allocating resources across cloud providers. It iterated through the list of cloud providers to find the first one with enough available resources to fulfill a user request.

Key Observations:

- **Efficient Resource Fulfillment:** In most cases, the system was able to allocate resources successfully across the cloud providers, showcasing the ability of the system to operate in a multi-cloud environment.
- **Limited Allocation Strategy:** The simple resource allocation mechanism does not consider the cloud provider's specific characteristics (e.g., available bandwidth, user priority, or failure state). More sophisticated strategies (e.g., considering priority or load) could improve resource allocation, especially when some providers are heavily loaded or have insufficient resources.

5. Scalability and Fault Tolerance

The simulation also demonstrated the scalability and fault tolerance of the intercloud system. The scaleResources method allowed each cloud provider to randomly scale up or down, simulating how cloud providers adjust to changing workloads. Additionally, the fault tolerance mechanism, where cloud providers can go down and later restore service, was crucial for ensuring the system's resilience. Table (3) shows the resource allocation when a cloud provider (e.g., CloudProviderA) fails. In this scenario, one can observe that requests R4 and R6 could not be allocated due to the failure of CloudProviderA.

Key Observations:

- **Scaling Behavior:** The scaling logic allowed cloud providers to dynamically adjust their resource pools. While this was a simplified mechanism (random scaling), it showed that the system could adapt to changing cloud resources. A more realistic scaling model could incorporate factors such as workload patterns, demand predictions, and cost-efficiency.

- **Failure Recovery:** When a cloud provider failed, it was marked as unavailable, and resources were not allocated to it. Once restored, it resumed operation. This demonstrated the system’s ability to tolerate failures without impacting the overall resource allocation process, enhancing the system's reliability.

Table 3: **Cloud Provider Failure Scenario**

Request ID	**Requested Resources**	**Allocated to Cloud Provider**	**Status**
R1	100	CloudProviderA	Allocated
R2	50	CloudProviderB	Allocated
R3	120	CloudProviderC	Allocated
R4	30	CloudProviderA	Failed
R5	200	CloudProviderB	Allocated
R6	150	CloudProviderA	Failed
R7	80	CloudProviderC	Allocated
R8	60	CloudProviderB	Allocated
...	...	...	...
R50	100	CloudProviderC	Allocated

6. Data Transfer Simulation

The DataTransfer class simulated data movement between cloud providers. Although this was a basic simulation, it illustrated the potential for intercloud data migration and the need for careful resource planning and optimization to minimize data transfer costs and latency. Table (4) demonstrates how cloud providers handle multiple failures and recovery processes during the simulation.

Table 4: **Cloud Provider Failure and Recovery**

Request ID	Allocated to Cloud Provider	Status
R1	CloudProviderA	Active
R2	CloudProviderB	Active
R3	CloudProviderC	Active
R4	CloudProviderA	Failed
R5	CloudProviderB	Active
R6	CloudProviderA	Failed
R7	CloudProviderC	Active
R8	CloudProviderB	Active
R9	CloudProviderA	Failed
R10	CloudProviderC	Active
...	...	...
R50	CloudProviderB	Active
CloudProviderA Status Restored		Restored
CloudProviderB Status Restored		Restored

Key Observations:

- **Data Transfer Latency:** The simulated data transfer introduced a delay, showcasing the potential impact of network latency on cloud operations. In a real-world system, factors such as bandwidth, network congestion, and data size would need to be considered when designing the intercloud exchange and optimizing data transfer routes.
- **Transfer Cost and Efficiency:** Data transfer between cloud providers can incur significant costs. An optimization mechanism that selects the most efficient path for data transfer (considering bandwidth, cost, and latency) could reduce operational expenses in real-world systems.

Conclusion and Recommendations

The intercloud simulation successfully demonstrated the viability of an intercloud resource management system that balances user requests, handles cloud failures, and simulates dynamic resource scaling. The system was able to efficiently allocate resources and ensure high availability by using load balancing and fault tolerance techniques.

However, several challenges remain. For instance, while the load balancing algorithm performed well in most scenarios; further optimization could be done to account for factors such as network latency, storage capacity, and regional availability. Additionally, the fault tolerance mechanism could be extended to simulate more complex failure scenarios, such as network partitions or partial cloud outages.

However, there are several areas where the system could be enhanced:

- **Advanced Load Balancing:** Implementing weighted load balancing or other intelligent algorithms based on resource availability, user priority, and provider capacity could improve efficiency.
- **Resource Allocation Strategy:** A more complex allocation strategy that considers provider state, resource utilization, and user priority could further optimize performance.
- **Realistic Scaling and Fault Tolerance:** Introducing more realistic scaling behaviors and fault tolerance mechanisms could enhance the system's adaptability to real-world cloud environment

Future Work

Future work will focus on enhancing the simulation by incorporating the following features:

1. **Advanced Load Balancing**: Investigate more sophisticated load balancing techniques, such as multi-objective optimization, to consider factors like network latency, regional availability, and resource usage patterns.
2. **Cloud Resource Heterogeneity**: Model cloud providers with heterogeneous resources, such as different types of virtual machines or storage solutions, to better reflect real-world multi-cloud environments.

3. **Scalability and Performance**: Evaluate the performance of the simulation under more extensive workloads, including thousands of concurrent user requests and the scaling of cloud providers.
4. **Energy Efficiency**: Incorporate energy consumption models to evaluate the environmental impact of cloud resource allocation and load balancing strategies.
5. **Integration with Real-World Systems**: Extend the simulation to integrate with real-world cloud environments, providing a more realistic and practical approach to Intercloud modeling.

By incorporating these improvements, the simulation can evolve into a comprehensive tool for research and optimization in Intercloud environments.

References

[1] Qian, L., Luo, Z., Du, Y., Guo, L. (2009). Cloud Computing: An Overview. In: Jaatun, M.G., Zhao, G., Rong, C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10665-1_63

[2] M. N. O. Sadiku, S. M. Musa and O. D. Momoh, "Cloud Computing: Opportunities and Challenges," in *IEEE Potentials*, vol. 33, no. 1, pp. 34-36, Jan.-Feb. 2014, doi: 10.1109/MPOT.2013.2279684.

[3] Adel Nadjaran Toosi, Rodrigo N. Calheiros,and Rajkumar Buyya, Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey, Vol. 47, No. 1,(2014)

[4] Yang, Y., He, Z., & Liu, D. (2012). Hybrid cloud computing architecture for resource allocation using multi-agent system. *International Journal of Cloud Computing and Services Science*.

[5] Sahu, S., Bhoi, A., & Awasthi, L. (2017). Load balancing in multi-cloud computing using dynamic resource allocation. *Journal of Cloud Computing*.

[6] Ranjan, D., Kato, H., & De Moura, M. (2020). Fault-tolerant architectures for Intercloud environments. *IEEE Transactions on Cloud Computing*.

[7] E. Emary, Hossam M. Zawbaa, Crina Grosan, and Abul Ella Hassenian: Feature Subset Selection Approach by Grey-Wolf Optimization. Industrial Advancement, Springer International Publishing,Vol. 63.2015 pp. 1-13

[8] R.C. Eberhart and J.Kennedy: A New Optimizer Using Particle Swarm Theory. Micro Machine and Human Science, Vol. 1, 1995, pp. 39-43

[9] Mousavi, Seyedmajid & Mosavi, Amir & Varkonyi-Koczy, Annamaria & Fazekas, Gabor. (2017). Dynamic Resource Allocation in Cloud Computing. Acta Polytechnica Hungarica. 14.