PAC1: Introduction and General Concepts

Sistemes Distribuïts

Estudis de Grau d'Enginyeria Informàtica

Semestre Tardor 2020

Índex de continguts

1

Transparency is the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. Transparency hides and renders anonymous the resources that are not of direct relevance to the task in hand for users and application programmers.

Access transparency enables local and remote resources to be accessed using identical operations. As an example consider a graphical user interface with folders, which is the same whether the files inside the folder are local or remote. Another example is an API for files that uses the same operations to access both local and remote files. From a user's point of view, access to a remote service such as a printer should be identical with access to a local printer. From a programmer‘s point of view, the access method to a remote object may be identical to access a local object of the same class.

Scaling transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms. The best-distributed system example implementing scaling transparency is the World Wide Web.

References

Coulouris, GF. [et al.]. Distributed Systems: Concepts and Design, 5th Edition. Harlow [etc.] : Addison-Wesley/Pearson Education, cop. 2012. 1063 p.

Sarmin FJ, Rashid M. Various Types of Transparencies in Distributed Homogeneous and Heterogeneous Database Systems International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-7 Issue-4, April 2018

2

Tiering is a technique to organize functionality into appropriate servers and, as a secondary consideration, on to physical nodes. This technique is most commonly associated with the organization of applications and services, but it also applies to all layers of a distributed systems architecture.

To understand the three-tiered client server architecture model it is important to consider first the usual functional decomposition of an application. Applications usually are decomposed in three parts:

1. the **presentation logic**, which is concerned with handling user interaction and updating the view of the application as presented to the user;

2. the **application logic**, which is concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is not limited only to business applications);

3. the **data logic**, which is concerned with the persistent storage of the application, typically in a database management system.

Now, let us consider the implementation of such an application using client-server technology.

In the **three-tier solution**, there is a one-to-one mapping from logical elements to physical servers and hence, for example, the application logic is held in one place, which in turn can enhance maintainability of the software. Each tier also has a well-defined role; for example, the third tier is simply a database offering a (potentially standardized) relational service interface. The first tier can also be a simple user interface allowing intrinsic support for thin clients. The drawbacks are the added complexity of managing three servers and also the added network traffic and latency associated with each operation.

In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:

1. A front-end web server serving static content, and potentially some cached dynamic content. In web-based application, front end is the content rendered by the browser. The content may be static or generated dynamically.

2. A middle dynamic content processing and generation level application server (e.g., Symfony, Spring, ASP.NET, Django, Rails, Node.js).

3. A back-end database or data store, comprising both data sets and the database management system software that manages and provides access to the data.

In the **two-tier solution**, the three aspects mentioned above must be partitioned into two processes, the client and the server. This is most commonly done by splitting the application logic, with some residing in the client and the remainder in the server (although other solutions are also possible). The advantage of this scheme is low latency in terms of interaction, with only one exchange of messages to invoke an operation. The disadvantage is the splitting of application logic across a process boundary, with the consequent restriction on which parts of the logic can be directly invoked from which other part.

The **LAMP stack** (Linux, Apache, MySQL, PHP/Perl/Python), is a common example of a two-tier web application. It is a very common example of a web service stack, named as an acronym of the names of its original four open-source components: the Linux operating system, the Apache HTTP Server, the MySQL relational database management system (RDBMS), and the PHP programming language. The LAMP components are largely interchangeable and not limited to the original selection. As a solution stack, LAMP is suitable for building dynamic web sites and web applications.

BASIS OF COMPARISON	TWO-TIER ARCHITECTURE	THREE-TIER ARCHITECTURE
Description	In 2-tier, the application logic is either buried inside the user interface on the client or within the database on the server (or both).	In 3-tier, the application logic or process resides in the middle-tier, it is separated from the data and the user interface.
Nature	2-tier architecture is client server application.	3-tier architecture is web based application.
Layers	Two-tier architecture is divided into two layers: Client Application (Client Tier) and Database (Data Tier).	Three layers in three tier architecture are: Client layer, Business layer and Data layer.
Intermediate	Direct communication takes place between client and server.	There is an intermediate between client and server known as application server.
Scalability	2-tier systems are less scalable and robust.	3-tier systems are more scalable and robust, as requests can be load balanced between servers.
Security	2-tier is less secured as client can engage the database directly.	Highly secured as client is not allowed to engage the database directly.
Number Of Servers	One server handles both application and database duties.	There are two separate servers available to handle applications and database separately.
Application Performance	In two tier architecture, application performance will be degraded upon increasing the users.	In three tier architecture application performance is good.
Deployment Costs	Deployment costs are high as no granularity exits.	Deployment costs are less due to builder tools.
Infrastructure	No extra infrastructure is required due to tight-coupling.	Infrastructure of middleware services can be used.
Reusability	Most clients are monolithic and thereby reusability not possible.	Reusability more with services implementation.
Hardware Flexibility	Hardware flexibility is very limited as only one client and one server exist.	Hardware flexibility is very robust with component-based environments to distribute services between 2nd and 3rd tiers.
Support For Internet	Support for internet is poor as bandwidth limitations may take more time for downloading big client programs.	Support for internet is good as services supporting downloading are distributed.
Server Failure	Can use backup servers in case of one server failure.	If one server fails, the services can be executed on other services.
Communication Process	For communication, client sends request to server and server directly responses to that client request.	For communication in this model, client establishes connection with application server first and then application server initiates second connection to the database server and vice-versa response sent back to client.
Communication Methods	Only connection-oriented methods calls are permitted.	Can support connectionless messaging, publish-subscribe and broadcast messaging.
Speed	2-tier architecture runs slow.	Three-tier architecture runs faster.
Maintenance	Complex to build and maintain.	Complex to build and maintain.
Flexibility	More flexible because it is loosely coupled.	More flexible because it is loosely coupled.

Table 1. Comparison of two-tier vs three-tier architecture

References

Coulouris, GF. [et al.]. Distributed Systems: Concepts and Design, 5th Edition. Harlow [etc.] : Addison-Wesley/Pearson Education, cop. 2012. 1063 p.

LAMP(software bundle).(2020). Retrieved October 8,2020 from https://en.wikipedia.org/wiki/LAMP_(software_bundle)

Multitier architecture. (2020) Retrieved October 8,2020 from https://en.wikipedia.org/wiki/Multitier_architecture

Difference Between Two-Tier And Three-Tier Database Architecture. Retrieved October 8, 2020 from : https://vivadifferences.com/difference-between-two-tier-and-three-tier-database-architecture/

3

Push-based communication is a style of Internet-based communication where the request for a given transaction is initiated by the publisher or central server. It is contrasted with **pull-based communication**, where the request for the transmission of information is initiated by the receiver or client.

We can find this communication models implemented in different protocols and services.

HTTP is mainly a **pull protocol** - someone loads information on a Web Server ans users use HTTP to pull the information from the server at their convenience. In particular, the TCP connection is initiated by the machine that wants to receive the file. Podcasting is also specifically a pull technology.

On the other hand, SMTP is primarily a **push protocol** - the sending mail server pushes the file to the receiving mail server. In particular, the TCP connection is initiated by the machine that wants to send the file. Synchronous conferencing and instant messaging are typical examples of push services. Other uses of push-enabled web applications include software updates distribution ("push updates"), market data distribution (stock tickers), online chat/messaging systems (webchat), auctions, online betting and gaming, sport results, monitoring consoles, and sensor network monitoring.

Both systems are also used in discovery services and replica management.

A pull-based approach is efficient when the read-to-update ratio is relatively low.

When comparing push-based and pull-based solutions, there are a number of trade-offs top be made. For example, we may consider a client-server system consisting of a single, nondistributed server, and a number of client processes, each having their own cache.

1. In push-based protocols the server needs to keep track of all client caches. Keeping track of all client caches may introduce a considerable overhead at the server.

2. The messages that need to be sent between client and the server also differ: In a push-based approach, the only communication is that the server sends updates to each client. In a pull-based approach, a client will have to poll the server, and if necessary, fetch the modified data.

3. The response time a the client is also different. In push-based approaches when the server pushes modified data to client caches, the response time at the client side is immediate whereas in pull-based there is a fetch-update time.

In conclusion, push-based communication seems to fit better in situations that the information must be available at the client immediately whereas pull-based communication allows the time uncoupling between the server and the client so the client can fetch the information when it suits to him.

References

Coulouris, GF. [et al.]. Distributed Systems: Concepts and Design, 5th Edition. Harlow [etc.] : Addison-Wesley/Pearson Education, cop. 2012. 1063 p.

Kurose, James F. Computer Networking: A Top-Down Approach, 7th Edition. Boston [etc.] : Addison-Wesley, cop. 2017. p 149

Tanenbaum AS. Van Steen M. Distributed systems: Principles and paradigms. 2nd edition. Pearson Education 2007.

Push Technology. Retrieved October 8, 2020 from https://en.wikipedia.org/wiki/Push_technology

Pull technology. Retrieved October 8, 2020 from https://en.wikipedia.org/wiki/Pull_technology

4

a)

The article deals with the challenge of scaling Web applications. The traditional transactional models are problematic when loads need to be spread across a large number of components. The author identifies two scaling strategies:

Vertical scaling: moving the application to larger computers

Horizontal scaling: performed along two vectors that can be applied at once.

Functional scaling: involves grouping data by function and spreading functional groups across databases.

Sharding: splitting data within functional areas across multiple databases.

One if the problems with Web applications is what is known as the **CAP theorem.** It is a conjecture about the possibility of ensuring **consistency**, **availability** an **partition tolerance** in Web services at once. This conjecture is pessimistic and affirms that a Web application can support, at most, only two of these properties with any database design.

Any horizontal scaling strategy is based in data partitioning; therefore designers are forced to decide between consistency and availability. **ACID** provides the consistency choice for partitioned databases whereas **BASE** provides the availability choice.

ACID transactions provide the following guarantees:

Atomicity. All of the operations in the transaction will complete, or none will.

Consistency. The database will be in a consistent state when the transaction begins and ends.

Isolation. The transaction will behave as if it is the only operation being performed upon the database.

Durability. Upon completion of the transaction, the operation will not be reversed.

On the other hand, **BASE** is the acronym of **basically available**, **soft state**, **eventually consistent**. The availability of BASE is achieved through supporting partial failures without total system failure. BASE accepts that the database consistency will be in a state of flux.

The auhors explain different ways to relax consistency in BASE. Consistency across functional groups is easier to relax than within functional groups. Different **decoupling strategies** between functional groups can be useful. Some of this decoupling strategies imply some kind of **indirect communication mechanism** like using message queues, and other techniques like using **timestamps**. Decoupling the operations and performing them in turn provides for improved availability and scale at the cost of consistency.

Finally, EDA (event-driven architecture) is presented as an architecture that can provide dramatic improvements in scalability and architectural decoupling.

b)

ACID approaches used to be simpler because they usually don't imply some kind of indirect communicaction. On the other hand, BASE imply using indirect communicaction mechanisms and ways to ensure the right order of transactions.

However, in terms of scalability, because of BASE uses decoupling mechanisms it is a better approach to scale Web applications than ACID. ACID systems tend to scale worse than BASE systems both from complexity and from performance points of view, but their trade-offs of consistency and availability are less polarized (i.e., a higher level of both concepts can be reached). Scaling systems to dramatic transaction rates requires a new way of thinking about managing resources. The traditional transactional models are problematic when loads need to be spread across a large number of components. Decoupling the operations and performing them in turn provides for improved availability and scale at the cost of consistency. BASE provides a model for thinking about this decoupling.

c)

Firstly, it is worth saying that ACID and BASE approaches can fit in different kinds of applications. BASE can be better in large distributed systems where temporal inconsistency can be tolerated whereas ACID may be better for systems that prioritize consistency over availability. For example, ACID may better fit in financial applications where consistency is crucial and there is a risk of system failure due to the non-synchronization of data nodes.

NoSQL refers to a set of database technologies that doesn't follow the relational model. NoSQL are non-relational databases, which are not structured in a table form. Most of the time, data is document-based with key-value pairs and does not have a schema. These databases can be structured in a more flexible form since we do not need to predefine a schema. The transactional model follows the BASE approach.

NewSQL databases are modern SQL databases that solve some of the major problems associated with traditional online transaction processing (OLTP) RDBMS. They seek to achieve the scalability and improved performance of NoSQL databases while maintaining the benefits of traditional database management systems and the ACID approach.

Navdeep Singh Gill provides a table that compares clasical SQL databases vs NoSQL and NewSQL.

Feature	SQL	NoSQL	NewSQL
Relational Property	Yes, follows relational modeling to a large extent.	No, it doesn't follow a relational model, it was designed to be entirely different from that.	Yes, since the relational model is equally essential for real-time analytics.
ACID	Yes, ACID properties are fundamental to their application	No, rather provides for CAP support	Yes, Acid properties are taken care of.
SQL	Support for SQL	No support for old SQL	Yes, proper support and even enhanced functionalities for Old SQL
OLTP	Inefficient for OLTP databases.	Supports such databases but it is not the best suited.	Fully functionally supports OLTP databases and is highly efficient
Scaling	Vertical scaling	Vertical + Horizontal scaling	Vertical + Horizontal scaling
Query Handling	Can handle simple queries with ease and fails when they get complex in nature	Better than SQL for processing complex queries	Highly efficient to process complex queries and smaller queries.
Distributed Databases	No	Yes	Yes

Table 2. Comparison of SQL, NoSQL and NewSQL databases

In conclusion, choosing between NoSQL and NewSQL has to do with the requirements of the application. However, NewSQL seems to be the most promising solution because it provides most of the benefits of clasical SQL relational databases plus those features that NoSQL provide for distributed database systems.

References

A comparison: SQL or NoSQL databases?. Retrieved October 8,2020 from https://www.educative.io/edpresso/a-comparison-sql-or-nosql-databases?utm_source=Google%20AdWords&aid=5082902844932096&utm_medium=cpc&utm_campaign=kb-dynamic-edpresso&gclid=Cj0KCQjwk8b7BRCaARIsAARRTL5QCYpyLLcluz0Uzx57lHIK5FsKN9nmxRY4LQKyjtfqJ-4-4HuZS3caArkJEALw_wcB

SQL vs NoSQL vs NewSQL: The Full Comparison.(2019). Retrieved October 8,2020 from https://www.xenonstack.com/blog/sql-vs-nosql-vs-newsql/

ACID or BASE databases? Consistency versus Scalability. (2018). Retrieved October 8,2020 from https://imelgrat.me/database/acid-base-databases-models/

5

a)

Transparency implies the concealment from the user and the application programmer of the complexities of distributed systems.

Full transparency implies that we can compile, debug and run unmodified single-machine code over effectively unlimited compute, storage and memory resources.

Some authors point out that there are limits to transparency because of latency, memory access, concurrency and partial failure. Although latency and memory access could be solved with future hardware implementations, these authors argue that concurrency and partial failures preclude the unification of local and remote computing. Distributed systems have no single point of resource allocation, synchronization or failure.

The main thesis of the authors of the paper is that **resource disaggregation** could be the definitive way to enable full transparency in the Cloud. The idea behind resource disaggregation is that all computing resources (compute, storage, memory) can be offered in a disaggregated way with unlimited flexible scaling.

To sustain this statement they present **two experiments** that analyze the trade-offs between performance and cost when comparing serverless and local computing resources. Monetary cost seems to be the most important drawback of the serverless functions in comparison to local threads.

The authors identify some **limits of disaggregation and transparency**. Compute disaggregation is just feasible when delays in job time can be tolerated. Access to networked storage is still slower than local memory access. Locality and affinity requirements still play a key role in stateful distributed applications. Finally, scaling transparency requires a new elastic programming model. Unchanged code based in a local programming model can't take advantage of the elasticity of disaggregated resources.

Finally, **five open research challenges** required to achieve full transparency for most applications are presented: 1) granular middleware and locality, 2) memory disaggregation, 3) virtualization, 4) elastic programming models, and 5) optimized deployment.

b)

The main advantage of the idea of full transparency is that the application programmer doesn't need to bother about things that do not have to do with the functionalities of the application that he/she wants to program. Therefore, he can focus his energy and skills in the main objectives of the application and be more productive. So it is easier for the user and the programmer because:

- they doesn’t have to bother with system topography. So it enables location transparency.
- they doesn’t have to know about changes. It enables scaling transparency.
- it is easier to understand. It enables access transparency.

The main drawbacks have to do with the loss of control. Failure transparency leads to concelament of different aspects that may be of interest in certain roles or circumstances.

- It might be more difficult to detect problems while programming.
- Optimization can not be done by programmer or user
- Strange behavior when the underlying system fails.
- Underlying system can be very complex.

c)

The article presents a quite optimistic perspective about the possibility of achieving full transparency in the near future thanks to resource disaggregation and serverless architectures. However, as Mike Roberts points out in his article about Serverless architectures there are significant trade-offs. One of the main problems is giving up control of the system to third-party vendors. This may lead to other problems such as lack of flexibility, cost changes, loss of functionality, unexpected limits or forced API upgrades. Besides, they are multitenant solutions and it may also lead to problems of optimization, security, robustness and performance.

References

Roberts M. (2018) Serverless architectures. Retrieved October 8, 2020 from https://martinfowler.com/articles/serverless.html#unpacking-faas

Janakiraman B.(2016). Serverless. Retrieved October 8, 2020 from https://martinfowler.com/bliki/Serverless.html