Crafting a suitable client/server architecture based on a performance characteristic

S. Ramesh Babu
Education and Research Department
Infosys Technologies Limited
3rd Cross Electronics City
Bangalore 561 229 India
rameshbabu@inf.com

Ramkumar Ramaswamy
Software Concept Laboratory
Infosys Technologies Limited
3rd Cross Electronics City
Bangalore 561 229 India
ramkumarr@inf.com

Abstract

Crafting a suitable client/server architecture involves myriad issues. Many established patterns of client/server architectures exist, and the process of crafting essentially involves choosing an appropriate pattern and adapting it to the application at hand. An important component in this decision process is the quantification of the relative performance of the application for the chosen pattern in a steady state. An important determinant of performance is the amount of network traffic that is generated by the client/server architecture. We consider the network traffic to be an indicator for the performance characteristic. We present a 6-step prescription for using this indicator to arrive at a client/server architecture, and use the TPC-C benchmark application as a running example to illustrate this process. Analysis of this application also provides sharp and revealing insights into the relative merits of various client/server patterns, and these insights are crucial to the selection of a pattern for a given application.

1 Introduction

The client/server paradigm allows business applications to adapt to changing requirements. The design of such systems should be a realistic and continuous process. One key activity in design is partitioning. Partitioning is the act of making cuts in the logical layered system structure (Figure 1) so as to yield a physical distribution of functionality. Effective partitioning requires a good understanding of the business and real-time requirements of the applications. There are several patterns for distributed business information systems [KlauKel97] based on decisions about where the application is partitioned (Figure 1). Real-life applications can be viewed as combinations of these patterns. For example, a 3-tier client/server architecture using client, application server and database is a combination of distributed presentation and distributed database patterns. The choice of an architecture, whether it is 2-tier or 3-tier, is driven partly by business considerations and partly by technical factors. Technical factors such as network traffic tend to be handled intuitively, but, as we will demonstrate, this can be quantified to yield useful estimates of the relative performance of different client/server cuts. In passing we point out that quantification may be difficult for greenfield projects, but gets easier with subsequent system evolution because there are fewer variables. For example, the structure of the database is much more stable after initial system deployment. Quantification is thus most suitable for re-engineering client/server development projects.

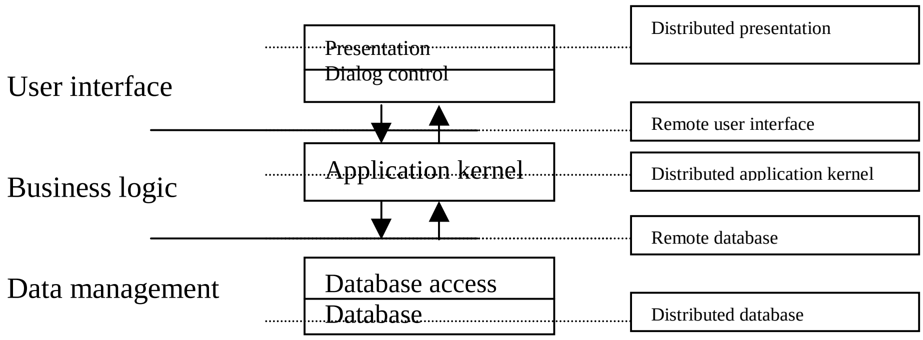

Figure 1. Client/server patterns resulting from cuts across layers

Figure 1 shows the following five patterns: i) Distributed presentation ii) Remote user interface iii) Distributed application kernel iv) Remote database v) Distributed database

We now focus on network traffic as an indicator of the suitability of a client/server pattern (cut). We present a 6-step prescription for using this indicator to arrive at a client/server architecture, and use the TPC-C benchmark application as a running example to illustrate this process. Though we have focussed on examining the client/server architectures, our focus is not on investigating any specific architecture but on presenting a general shop-usable process of analyzing and comparing client/server architectures. Our approach is to calculate the network traffic for a given pattern (remote user-interface, remote database etc.) and for a given type of application (centralized or distributed). Furthermore, note that there is also a small contribution of network traffic because of business considerations. For example, in a warehouse supplier application, a warehouse obtains about 10% of the items from other warehouses. We will exclude such traffic in our analysis because such an element of traffic is beyond the system architect's control, since it is an element of the way the business functions.

Firstly, some network traffic results are available in the literature such as [1]**.** The key contribution of [1] is that it has presented a catalog of prevalent patterns for distributed client/server systems. It also makes a comparison of the relative network traffic generated by each pattern. However, these comparisons are purely qualitative. For example, it is reported that network traffic in remote database architecture is high compared to remote user interface, but there is no indication of the factor involved. In this paper, we will quantify such results and make some inferences about the network traffic ratio. For example, we will make a *quantitative* comparison between the remote user interface and remote database patterns. These inferences are then used to aid the selection of one type of architecture over another.

The organization of the paper is as follows. In section 2, we outline the top-level steps in selecting and evaluating a client/server architectural pattern. In section 3, we first provide a quick overview of the TPC-C benchmark and then move on to illustrating each step outlined in section 2 in detail with all the necessary analysis. The last section presents conclusions and future work.

2 The steps

In this paper, we present a set of guidelines to select a suitable client/server architecture based on the following steps:

1. Identify a set of candidate patterns from Figure 1 based on business considerations
2. Identify a representative steady state for the application
3. Quantify the steady state network traffic between application layers
4. Identify the marshalling factor
5. Revisit the choice of candidate patterns based on network traffic analysis
6. Work out the details of the distribution pattern using the steady state information

Note that the last two steps may be iterative in nature, for example, if the complete database design is not available to the architect.

The characteristic feature of the approach is that it allows one to start with a candidate solution based on intuition, and then vets the intuition using numerical estimates. The main elements in the approach are (a) analysis by quantification rather than qualification and (b) comparison of analysis across different patterns.

3 Detailed description of the steps

Let us walk through each of the above steps using a reference application. It is important to note that the application must be well understood in terms of business lines. For example, it makes sense to identify what are the various types of transactions in an application and which transaction is triggered at what site. Since the domain of applications that we are interested are of On-line Transaction Processing (OLTP), we now consider TPC-C [4] as a representative example for the analysis. The TPC-C is a widely used benchmark for quantitative comparison of different OLTP systems. For this reason, it is also a favorite for modeling studies [3]. Before proceeding with the analysis, we provide a brief overview of this application (which we henceforth refer to as just "TPC-C").

The TPC-C is a representative of a typical order entry application which is an amalgamation of interactive and batch transactions. TPC-C benchmark is 10 times heavier than TPC-A and TPC-B benchmarks [4]. The application portrays the activity of a wholesale supplier that manages, sells and distributes a product using a network of warehouses situated in districts. The benchmark, which is centered on an order-entry application, includes five transactions and nine tables. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. The tables include the *Warehouse*s (W) and associated *District*s (D). Each Warehouse maintains *Stock*s (S) of the *Item*s (I) sold. Each district handles its *Customer* (C) information and the *New-orders* (NO) placed by the customers. Each district maintains the status of its *Order*s (O) and customer order *History* (H). The status of each item ordered is treated through an *Order-line* (OL) table.

3.1 Step 1. Identify a set of candidate patterns based on business considerations

The first step in the methodology is to identify a candidate pattern from the several patterns for distributed business information systems. Most of the client/server systems are derived from the layered architecture [1] shown in Figure 1. At this stage, this selection of patterns is motivated primarily by the business. For example, consider a warehousing system where the application is likely to have to deal with geographically distributed warehouses. Typically, in such an application, warehouses monitor their own inventory. Therefore, it is advisable to select a distributed database pattern that allows some data to be owned by respective warehouses. Similarly, consider another example of a remote banking system where the application is likely to have to deal with geographically distributed user interface.

3.2 Step 2. Identify a representative steady-state for the application

In this paper, our focus is on steady-state performance. A steady-state is a stable state of the system which includes both online and offline processing. For online processing, response times matter i.e. certain type of transactions should respond in fixed amount of time. For example, in TPC-C application, the new-order and payment transactions are to be executed online. For offline processing, time windows matter i.e. certain type of transactions must have been completed within a time frame. It doesn't matter when they get executed as long as they get executed within the time frame. For example, in TPC-C application, the delivery transaction is to be executed off-line. So, typically, off-line processing may have lower priority. The steady state of the TPC-C application comprises of a minimum percentage of mix: 43% payment, 4% order-status, 4% delivery, and 4% stock-level. Assume that there are 45% of new-order transactions[1].

---

[1] There is no minimum for the New-Order transaction as its measured rate is the reported throughput.

3.3 Step 3. Quantify the steady state network traffic between application layers

In this section we present how to calculate the network traffic and analyze the same. The analysis has two components:

a) analyze the network traffic for each transaction executing stand-alone
b) find the net network traffic by applying the relative weight of each transaction in the steady state to the results obtained in (a)

These components are addressed in §3.3.1 and §3.3.2. Since the terminal I/O in TPC-C has a form-based user interface, unlike the object-based interface, the dialog control of Figure 1 acts as a transfer function. Hence we will not consider the distributed presentation pattern because with dialog control being a transfer function, it is same as remote presentation. Similarly, the distributed application kernel pattern is suited for On-line Analytical processing (OLAP) such as decision support systems with object oriented interface. Since TPC-C is meant for OLTP, this distributed application pattern is also ruled out. We are thus left with a)remote user interface, b)remote database and c)distributed database. We will analyze the first two in this section, and will consider the distributed database pattern separately in a detailed way.

Note that the intra-layer or horizontal traffic, i.e. network traffic from a layer in one warehouse site to a similar layer in another warehouse site, is insignificant. For example, with an average of 10 items per order, approximately 90% of all orders can be supplied in full by stocks from the same warehouse. That means, other warehouses that fulfill the rest of the 10% don't contribute much to the network traffic.

In estimating the network traffic generated by a user action, it is wise to be conservative. Here are some simple rules to follow.

- If a user searches either by XX or by YY to get ZZ and XX generates more traffic then it is appropriate to take the estimate for XX. For example, in case of order-status transaction, there is a choice between the customer-id and the customer-name and we chose customer-name. This is because the length of the customer-name is more than the customer-id.

- In the figure 1, certain layers act as identity function while sending or receiving the bytes. For example, business logic layer acts as an identity function, while transferring the bytes, from presentation layer to data management layer.

- In case of transactions where the entries are selected based on a customer last name, we account for 3 records to be selected.

- Once a particular record is selected from a relation, it is available for future references in the business logic layer.

- Assume there is an initial screen that will identify the type of transaction i.e. whether it is a new-order, payment, order-status, delivery or stock-level transaction.

The general algorithm to calculate the network traffic is as follows:

Pseudo algorithm

```
For each transaction  (total 5 transactions)
        For each pattern (total 2 patterns)
                Calculate the byte flow
```

3.3.1 Analysis for a stand-alone transaction

We show the details of network traffic analysis for new-order transaction as follows. Refer to figure 1.

Each transaction generates requests and inserts/updates etc. as given in the standard specification of the TPC-C benchmark [4].

User interface to business logic layer: The attributes that are input into the terminal are D_ID, C_ID OL_I_ID, OL_SUPPLY_W_ID, OL_QUANTITY. The summation of the length of the bytes for each of these attributes gives the total bytes flow from user interface to business logic. This number is 18.

Business logic to data management layer: As far as the business logic is concerned the same number of bytes as above are transferred to the data management.

Data management to business logic layer: In case of a new order transaction, the relational tables that are accessed are Warehouse, District, Customer, Order, New-order, Item, Stock, Order-line.

Here is a table stating which transaction has an access (i.e. read, write, insert, update etc.) to what relations. The columns are the relations and the rows are the transactions.

Transaction/ Relation	W	D	C	NO	O	I	S	OL	H
New-order	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Payment	Yes	Yes	Yes						Yes
Order-status			Yes		Yes			Yes	
Delivery			Yes	Yes	Yes			Yes	
Stock-level		Yes				Yes	Yes		

Each access of a record in Warehouse, one access and one update of a record in District, one access of a record in Customer, one insert of a record in Order and New-order contributes to 966 bytes (89+2*95+ 655+24+ 8) [4]. Each item in the order, out of 10 orders, one access of Item, Stock, one update in Stock, one insert in Order-line contributes to 748 bytes (82+2*306+54). On an average, the 10 orders contribute 7480 bytes. Therefore, the net result is 8446 bytes (966 + 7480).

Business logic to user interface layer: The attributes that are required for user interface are the ones that are required for display. These are a) repeat group category that contribute 10*54 bytes b) non-repeat group category which contribute 87 bytes. Therefore the net result is 627 bytes.

A similar calculation for other transactions yields the following table.

Item/Transaction	New order (45%)	Payment (43%)	Order status (4%)	Delivery (4%)	Stock level (4%)	Steady state
User interface to Business logic	18	30	18	6	8	34
Business logic to Data management	18	30	18	6	8	34
Data management to business logic	8446	1724	2529	2454	11201	11015
Business logic to User interface	627	543	1120	30	11	980

Table 1. Network traffic flow across various layers

Note the rightmost column of the table describes the steady-state network flow which is obtained by the weighted average of the respective entries in that row. In the steady-state, for example, there are about 34 bytes of flow between business logic to data management as against 11015 between data management to business logic.

Let us now consider what happens with a ***remote user interface*** and with a ***remote database.*** Row 1 and row 4 in Table 1 contribute to the network traffic. In case of remote database, row 2 and row 3 in Table 1 contribute to the network traffic. Taking these into considerations, we will now calculate the ratio of network traffic for remote database versus remote user interface.

Item/Transaction	New order	Payment	Order status	Delivery	Stock level
Remote database / Remote user interface (approx.)	235	37	101	207	700

3.3.2 Steady state analysis

While the previous section focuses on transactions being executed in isolation, we now show what happens when a transaction mix is executing in a steady state. To achieve this, we now weight these numbers using the steady state specification of the TPC-C.

Weighted average for all transactions in remote database pattern:

As per the specification, the minimum percentages of mix of transactions are 43% for payment, 4% order-status, 4% delivery, and 4% stock-level. Assume that there are 45% of new-order transactions. Then from Table 1 we have

(45*(8446+627) + 43 * (1724 + 543) + 4 * (2529+ 1120) + 4 * (2454+ 30) + 4 * (11201+ 11)) / 100
= 5751 (approx.)

This quantity gives the average network traffic in the steady state for remote database architecture when all transactions are running concurrently.

Weighted average for all transactions in remote user interface pattern:

From Table 1, we have :

(45 * (18 + 18) + 43 * (30 + 30) + 4 * (18 + 18) + 4 * (6 + 6) + 4 * (8 + 8)) / 100
= (45 * 36 + 43 * 60 + 4 * 36 + 4 * 12 + 4 * 16)/ 100
= 45 (approx.)

This quantity gives the average network traffic in the steady state condition for remote user interface architecture when all transactions are running concurrently.

From both the above quantities, we can infer that *for an order-entry application the remote user interface architecture is at least 127 times better than remote database architecture*!

3.4 Step 4. Identify the marshalling factor

There is another element contributing to the network traffic. This factor is due to marshalling and its role is prominent in a distributed computing environment. Recall that marshalling is the packaging of intelligible data, such as size and type of argument of a function, into raw stream to be sent over the network to the server [2]. The marshalling factor, though significant, is highly dependent on the development environment, platform and tools. For example, consider in a large 1000 person-month distributed-object application that we developed for a leading US insurance firm for about 8000 users spread over 70 different locations with nearly 5 million transactions per year. Visual Basic was used as front-end, MTS and DCOM was used as middleware and Oracle as database on NT/Unix server. It was observed that the marshaling factor was approximately 20. In our example, however, we do not include a marshalling factor.

3.5 Step 5. Revisit the choice of candidate patterns based on network traffic analysis

Based on the numbers obtained so far, it is apparent that remote database has high network flow. We will now see how we can cut down the network traffic flow by distributing the database. We need to work out details of distributed database design using the same steady state information.

3.6 Step 6. Work out the details of database distribution

Our efforts in this final step will be towards investigating how the network flow for TPC-C benchmark can decide the distribution of relations in a distributed database pattern. We will assume there are k, $k \geq 2$, warehouses.

We want to distribute nine relations in such a way that the distribution takes the advantage of localization of processing and thus not affecting to the network traffic. The relations are well placed in such a way that these relations are related in terms of business lines.

One of the main issues concerning the placement of relations is what happens when a relation R_1 "wants to be" near both R_2 and R_3 and these two are at different sites. One of our goals is to show how judicious allocation of tables to sites in a distributed design can minimize the network traffic. This requires us to

- Understand which transaction should be fired from which place
- Understand which relation should be placed at which place

Consider a scenario where there are two sites
a) Customer service site (S1)
b) Single warehouse site (S2)

Let us identify which, out of five transactions, are the ones that are triggered at site S1. If we notice carefully based on the intuition from business lines, the transactions new order, payment, order status, and delivery are triggered at the site S1. Next, the only transaction that makes sense to have it at site S2 is the stock level transaction. Given this configuration, let us try identifying whether a relation should be placed in S1 or at S2 based on the calculation of the network traffic.

Consider the placement of Stock relation at S1. If a Stock level transaction is fired (from S2), then this will involve a network traffic of a 306 bytes (i.e. typical row length in Stock relation) [4]. Under steady state, with 4% mix of stock-level, this yields in a network traffic of 1224 (4*306) bytes. Similarly, if the Stock relation is placed at S2 then, because of new-order transaction, this will involve a network traffic of 3060 (306*10) bytes. Under steady state, with 45% mix of new-order transaction, this yields a network traffic of 137700 (45 * 3060) bytes. Clearly, placing Stock relation at S1 is better than placing it at S2. Note that this calculation doesn't involve the network traffic across relations.

Now, consider
Relations at S1 site only: Customer, New order, Order, Stock and History
Relations at S2 only: Warehouse, District and Item.

As per the TPC-C specification, the relation Stock can be either at site S1 or at site S2. Since Stock is functionally closer to the activities of warehouse site, let us place it at site S2. Furthermore, some of the relations can be placed at both S1 and S2. We will now see how network traffic influences the placement of the Order-line.

Pseudo algorithm

For each transaction (Total of 5)
For the distributed database pattern
Calculate the network
a) Order-line is located at S1
b) Order-line is located at S2

1. New-order transaction

The new-order transaction is fired at S1. The relations Customer, New-order, Order, Stock, History are located at S1 and Warehouse, District and Item are located at S2.

a) Consider the case when Order-line is located at S1.

Since the relations Warehouse, District and Items are located at S2, an access to each of these relations will involve network traffic as follows:

i) Warehouse record = 89 bytes
ii) District record = 95 bytes
iii) Item record = 10 * 82 bytes

So, the total is 1004 bytes.

b) Consider the case when Order-line is at S2.

Since Order-line is located at S2, an access to each of these relations will involve additional network traffic due to Order-line. This quantity is 10 * 54 = 540, in addition to 1004, leading to a total of 1544 bytes.

The rest of the cases can be calculated on similar lines.

Transaction	New-Order	Payment	Order-Status	Delivery	Stock
Order-line at S1	1004	NA	0	0	360
Order-line at S2	1544	NA	540	540	306

Note that NA stands for not applicable. Since payment transaction is not dependent on Order-line, the placement of Order-line either at S1 or at S2 will not affect the network traffic. Next, consider the order-status transaction that is fired at S1. Since, order- line is at S1 there will be no network traffic. Consider delivery transaction and order-line placed at S1. Since all the relations are available at S1, there will be no network traffic. Having Order-line at S1 leads to an average network traffic of 46620 (45* 1044 + 4* 360). Having Order-line at S2 leads to an average network traffic of 75024 (45 * 1544 + 4 * 0 + 4 * 540 + 4 * 540 + 4 * 306)

This analysis shows that it is advantageous to have Order-line placed at S1 than S2 because it cuts down the network traffic by a factor of 1.6.

4 Conclusion

In this paper, we have presented a 6-step prescription for selecting a suitable client/server architecture. We have used the popular TPC-C benchmark application to illustrate each step, and, in the process of analysis, quantified the relationship between several well-known client/server architectural patterns. Based on the analysis, we have presented both the average network traffic and weighted average network traffic. We have also considered and analyzed a scenario where the network traffic affects the distribution of relations. We believe that this approach can be applied to measure results for other benchmarks too.

Though the network traffic is an important determinant of client/server performance, it forms only one component of the most real-life client/server architecture decisions. Other factors that contribute to performance are the impact of object-oriented design, impact of graphical user interface, performance of database technology, and of course cost. A more comprehensive decision process would have to take all these factors into consideration. Future work should address this.

References

[1] Client/Server Architectures for business information systems: A pattern language, A publication in a conference, In Proceedings of PloP Conference, 1997.

[2] Learning DCOM: T. L.Thai, A Book, O'Reilly, 1999.

[3] Leutengger, S. T. and Dias D.: A modeling study of the TPC-C benchmark, A publication in a conference, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, May 1993.

[4] TPC-C Benchmark C Standard Specification, August 25, 1998