8-25-1995

Comparing Coding, Testing, and Migration Costs for Two and Three Tier Client/Server Architectures

John Gallaugher
Syracuse University

Comparing Coding, Testing, and Migration Costs for Two and Three Tier Client/Server Architectures

John Gallaugher, Syracuse University

Introduction

One can consider most end-user applications as consisting of three components: presentation, logic, and data. An architecture for a distributed, client/server system can be classified in terms of how these functions are split between software entities and where functions are located on a network. Although a virtually infinite number of possibilities exist for distributing and linking application components, research suggests most contemporary architectures can be defined in terms of a limited subset of alternatives. This paper focuses on issues related to the distributed logic scenario for two-tier and three-tier client/server architectures popular in implementing database transaction-oriented distributed systems.

Organizations with portfolios of applications containing common and possibly reusable components would be interested in identifying the costs associated with coding, testing, and migrating (CTM) under different distributed architectures. Recent articles in practitioner literature have discussed the relative advantages and selection heuristics for these architectures,,,. However, these have not operationalized cost justification for choosing one alternative over the other. This paper begins to address this issue by offering formulations to assess CTM costs under specific scenarios. Research is underway to estimate the impact of this model on organizational decision making with regard to architecture choice.

Two Tier Architecture

The two-tier, distributed client/server architecture divides presentation, processing, and data into two distinct units - a client which executes on a PC, and a database server which executes on another node of the network. Although some systems make it possible to populate the DBMS with code, this paper will be concerned primarily with that code stored on the client (workstation) side.

Three Tier Architecture

Three-tier architecture splits user interface, functionality, and data into three distinct units or tiers. In the three tier scenario, the amount of logic in the PC clients is minimized . When these clients evoke a business rule or access data, they send a request to a middle

tier server. The middle tier then either fulfills the request itself, sending the result back to the PC, or more commonly, if additional resources are needed (such as data or the calculations of another server), the middle tier server acts as a client to an additional server. Three tier architectures are typically implemented using supporting technologies such as Open Software Foundation's Distributed Computing Environment (OSF/DCE) and Sun/USL's Open Network Computing. Further discussion of resource coordination mechanisms for distributed logic can be found in and .

CASE I - Response to a Business Rule Change

Consider the case in which the business logic of an organization changes, requiring modification of code in an organization's application portfolio.

Example 1 - Rule Modification Requires No Parameter Changes

In Example 1 we will assume the logic of a business rule, R1*, changes, but the parameters (inputs and outputs) required to invoke the rule do not change.

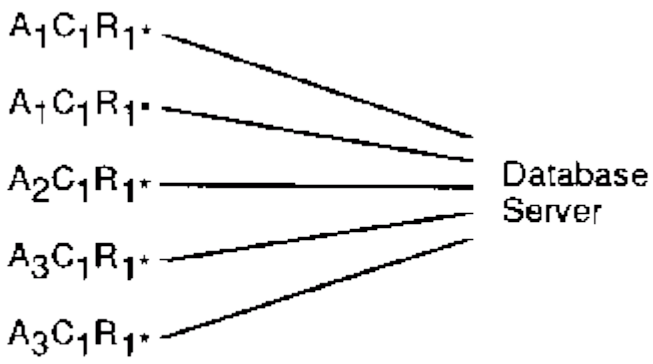

Fig. 1: Two-Tier Apps. under Ex. 1

Figure 1 presents the case of a two-tier architecture where three distinct applications, A1 to A3 (such as quoting, order fulfillment, and customer return processing) contain the bulk of the application code and are affected by a change in business logic. This change is depicted as altering business rule R1*. In this scenario the change in business rule R1* must be coded in three distinct applications. Since the exchange parameters of the rule do not change, the calling mechanism (procedure call or C1) required to invoke the rule does not need to be changed. Each application must be tested separately and the applications must be migrated to the storage devices for each client workstation (five migrations in the case above).

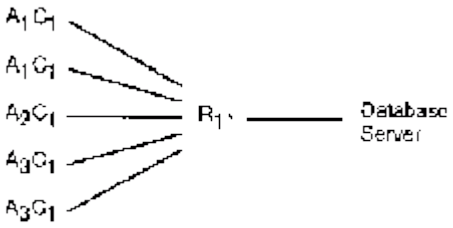

Fig. 2: Three-Tier Apps. under Ex. 1

Now consider Example 1 constructed in a three-tier architecture where the business logic for Rule R1* has been split into a separate software unit executing in a distinct middle tier (Figure 2). Each of the client applications access the common logic of R1* by executing a Remote Procedure Call C1. The change to R1* now occurs at a single location. Since the calling and receiving parameters of the business rule do not change, no changes need to be made in the client applications A1 - A3. Testing and migration for this code is similarly centralized, drastically reducing the complexity and cost of executing the change.

The values in curly brackets in Table 1 shows the maintenance iterations required for the changes in Example 1 for the two-tier vs. three-tier architecture. The advantage for Example 1 in terms of iterations lies with the three-tier environment, however coding, testing, and migration weights vary from two and three tiers. More general rules assessing the costs associated with coding, testing, and migration are shown in Table 1 and were created by considering the following variables:

n - number of client applications containing the rule being changed
f - number of middle-tier functionality servers containing the rule being changed
d2 - cost to code the rule change in the workstation client
d3 - cost to code the rule change in the middle tier server
t2 - testing cost for the workstation component
t3 - testing cost for the middle-tier component
cd - cost to code a call to a business rule
c - number of storage devices (workstation hard disks or LAN-based application servers) containing applications A1 through An
s - number of storage devices containing the logic for middle-tier rule
cm - cost to migrate a single client
fm - cost to migrate a single middle-tier server

	Two Tier	Three Tier
Coding	$\sum_{j=1}^{n} d2_j$ {3}	$\sum_{k=1}^{f} d3_k$ {1}
Testing	$\sum_{j=1}^{n} t2_j$ {3}	$\sum_{k=1}^{f} t3_k$ {1}
Migration	c * cm {5}	s * fm {1}

Table 1: CTM Costs & [iterations] under Ex. 1

In this scenario, the cost advantage may lie with the three tier environment. This advantage would be most clear in cases where the number of applications affected by a change and the number of client storage devices requiring migration are large.

Example 2 - Rule Modification Requires Parameter Changes

Example 2 is similar to Example 1, however assume the change to business rule R1* requires a change to the rule's calling parameters (i.e. the addition of a variable or a change to a data type). This requires all calls to R1* (depicted in the figures as procedure call C1*) to be modified. Figure 3 depicts the two tier case for Example 2, Figure 4 shows the three tier case.

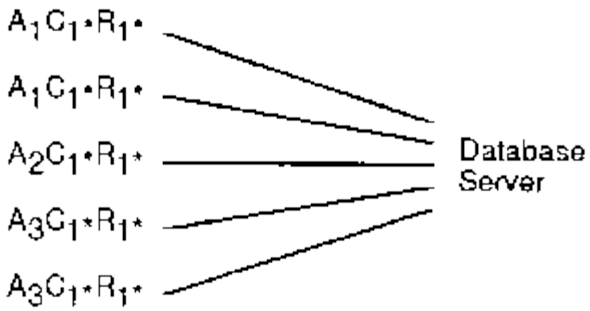

Fig. 3: Two-Tier Apps. under Ex. 2

The changes in C1* indicate that the workstation component of each application must be modified for both architectures. The three-tier architecture additionally requires one to alter, test, and migrate the middle-tier logic for rule R1*. While Example 1 indicated that the three-tier architecture had fewer maintenance iterations, the values in brackets in Table 2 show that the iteration advantage for Example 2 lies with the two-tier architecture.

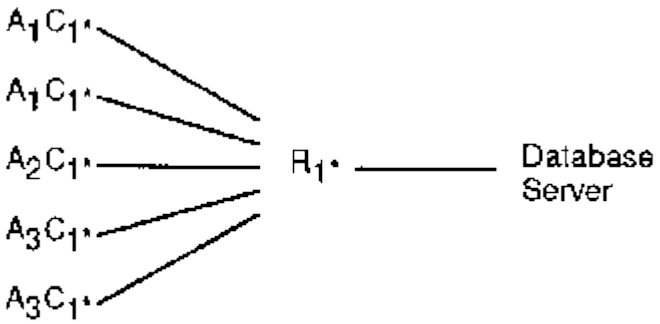

Fig. 4: Three-Tier Apps. under Ex. 2

Table 2 also shows formulae for calculating CTM costs under Example 2. For simplicity, constant costs which are roughly equal in terms of costs and iterations between two and three tier systems (such as database and user interface changes) are dropped from the comparison equations. If one eliminates constant terms among the two environments it becomes clear that the three-tier environment will always have more CTM iterations and greater testing and migration costs in a scenario where calling parameters change. Coding costs may also be greater, particularly in cases where d3 > d2, n is small and/or f is large.

	Two Tier	Three Tier
Coding	$\sum_{j=1}^{n}(d2_j + cd_j)$ {3}	$\sum_{j=1}^{n}(cd_j) + \sum_{k=1}^{f}(d3_k)$ {4}
Testing	$\sum_{j=1}^{n}(t2_j)$ {3}	$\sum_{j=1}^{n}(t2_j) + \sum_{k=1}^{f}(t3_k)$ {4}
Migration	c * cm {5}	(c * cm)+ (s * fm) {6}

Table 2: Maintenance Costs for Two vs. Three Tier

CASE II - Changing the Workstation Client Development Tool

Gauging costs associated with migrating to a new tool is critically important given the level of increased market instability and rapid innovation in the PC tool market. Current client development tools are universally proprietary and as such, any code written for one client tool can not be reused when migrating to another.

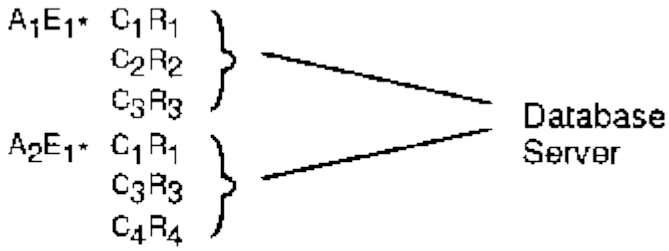

Fig. 5: Client tool migration for Two-Tier Apps.

Figure 5 illustrates an example where two applications are migrated from one development environment to another (E1*). In this case the entire investment in client-side code must be scrapped and re-written in the new environment Code which must be re-written is identified in italics.

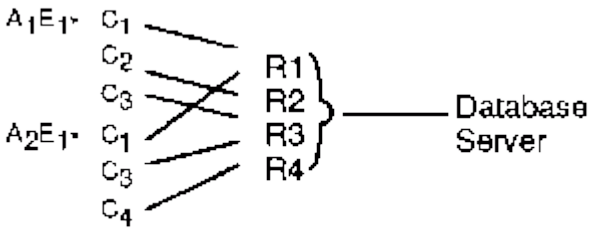

Fig. 6: Client tool migration for Three-Tier Apps.

Figure 6 demonstrates the savings achieved in changing client development environments under a three tier architecture. In this scenario business rules are coded in a middle tier which can be re-used when the client environment is switched, greatly reducing the burden of re-development.

	Two Tier	Three Tier
Coding	$\sum_{j=1}^{N}[\sum_{k=1}^{r}(d2_k)+\sum_{l=1}^{c}(cd_l)]$	$\sum_{j=1}^{N}(\sum_{l=1}^{c}cd_l)$
Testing	$\sum_{j=1}^{N}[\sum_{k=1}^{r}(t2_k)+\sum_{l=1}^{c}(ct_l)]$	$\sum_{j=1}^{N}(\sum_{l=1}^{c}ct_l)$

Table 3: Maintenance Cost Formulae

Table 3 lists cost formulae for comparing the re-development effort associated with migrating to a new client tool under two vs. three tier architectures. The final deployment of code on PC hard disks (migration) is not depicted, as these costs will be the same in both two and three tier applications. Variables which were not defined earlier are defined below.

r - number of business rules used by an application
c- number of times business rules are called by an application
N - number of applications whose platforms are being switched
ct - cost to test a call to a business rule

Given the equations in Table 5 it can be said that changing the client development environment will always cost

$$\sum_{j=1}^{N}[\sum_{k=1}^{r}(d2_k + t2_k)]$$

more in a two tier environment than in a three tier environment.

Conclusions and Ongoing Research

Scenarios were presented for maintenance and platform (tool) switching which compared costs related to coding, testing, and migration (CTM costs). In the case where maintenance to code logic is required, but request/response calling parameters of the rule do not change, the three tier system may have cost savings over two tier systems. These savings are greatest in cases where the number of applications affected by a change is large, where updated code must be migrated to many workstation storage devices, and where the difference in coding and testing costs between two and three tier applications is small. Three tier architectures always have higher testing and migration costs than two tier systems when the business rule change requires a change in calling parameters. CTM costs are also always less for three tier systems when comparing the cost of switching from one proprietary client development tool to another.

CTM costs are only part of the total cost equation. A number of issues were not examined including networking costs, hardware costs, response time and fault tolerance. Three tier architectures can also facilitate the integration of disparate systems and can prevent islands of information from forming within an organization - critical issues not fully explored in the models presented. These issues can be equally significant in arriving at an appropriate architectural choice for a given distributed application. Opportunities for future research exist in testing the validity of equations presented in this paper, creating new formulas incorporating additional costs cited, and examining opportunities for savings in code reuse/sharing for three-tier architectures.