PART I

Computing platforms

Barely 50 years after the birth of enterprise computing, cloud computing promises to transform computing into a utility delivered over the internet. A historical perspective is instructive in order to properly evaluate the impact of cloud computing, as well as learn the right lessons from the past. We first trace the history of enterprise computing from the early mainframes, to client-server computing and 3-tier architectures. Next we examine how the internet evolved into a computing platform for enterprise applications, naturally leading to Software as a Service and culminating (so far) in what we are now calling cloud computing. Finally we describe how the 'enterprise architecture' function within IT departments has evolved over time, playing a critical role in managing transitions to new technologies, such as cloud computing.

CHAPTER 1

Enterprise computing: a retrospective

1.1 INTRODUCTION

By ‘enterprise computing’ we mean the use of computers for data processing in large organizations, also referred to as ‘information systems’ (IS), or even ‘information technology’ (IT) in general. The use of computers for enterprise data processing began in the 60s with the early mainframe computers. Over the years enterprise computing paradigms have changed dramatically with the emergence of new technology: The advent of the PC in the 80s led to the replacement of large mainframe computers by client-server systems. The rise of the internet in the 90s saw the client-server model give way to web-based enterprise applications and customer-facing e-commerce platforms.

With each of these advances, enterprise systems have dramatically improved in terms of scale and ubiquity of access. At the same time their complexity, and consequently cost, has increased as well: Trillions of dollars are spent world-wide on information technology, including hardware and software purchases as well as application development (in-house or outsourced). It is also estimated that enterprises spend between two and ten percent of their revenues on IT.[1]

[1] From Gartner reports.

Now, cloud computing offers the potential for revolutionizing enterprise computing once more, this time by transforming computing itself into a utility that can be accessed over the internet. In his recent book *The Big Switch* [8], Nicholas Carr compares the possible ramifications of such a change to the creation of the electricity grid in the early twentieth century. Before the grid, industries built and ran their own power generation plants, much as enterprises deploy and manage their own computing systems today. After the grid came along, by 1930, 90 percent of electricity in the US was produced by specialized power utilities and delivered to consumers over power lines [8]. Barely 50 years had elapsed since Edison's invention of a reliable incandescent light-bulb. Will there be a similar revolution in enterprise computing, 50 years after its birth? Only time will tell.

The key elements of cloud computing, as we see it today, are: (a) computing resources packaged as a commodity and made available over the internet, (b) the ability for end-users to to rapidly provision the resources they need and (c) a pricing model that charges consumers only for those cloud resources they actually use. Further, as a result of centralization of computing, significant economies of scale can be exploited by a cloud provider and passed on to enterprise IT. Not surprisingly, much of the interest in cloud computing today is based on expectations of such cost savings. Finally, the concentration of massive clusters of computing resources within cloud providers opens up possibilities for large-scale data analysis at scales unheard of until now. In the process a number of new programming models and development tools have been developed, both to enable large-scale computations as well as dramatically improve software development productivity, and these also fall within the purview of cloud computing.

In this book we shall delve into the technical details of all the above elements of cloud computing: The major cloud platforms are covered in Chapter 5. Chapter 6 examines the potential cost savings from cloud computing. Key technologies essential for building cloud platforms are covered in Chapters 7, 8 and 9. New programming models and development paradigms are the subject of Chapters 10, 11 and 12. The impact of cloud computing on many of the essential aspects of enterprise computing, from data models to transaction processing, workflow and analytics, is examined in Chapters 13, 14, 15 and 16. Chapter 17 presents a snapshot of the cloud computing ecosystem, as it stands today. Finally we conclude, in Chapter 18, by discussing how enterprise IT is likely to adopt cloud computing in the near future, as well as speculate on the future of cloud computing itself.

However, before embarking on this journey, we first revisit the history of enterprise computing architectures in Chapters 1, 2 and 3. As we shall see, in many ways we have come full circle: We began with large centralized computers and went through phases of distributed computing architectures, saw the reemergence of a centralized paradigm along with the emergence of the internet as a computing platform, culminating (so far) in what we now call cloud computing.

1.2 Mainframe architecture

We can trace the history of enterprise computing to the advent of 'third-generation' computers in the 60s; these used integrated circuits as opposed to vacuum tubes, beginning with the IBM System/360 'mainframe' computer and its successors, which continue to be used to date, e.g. the IBM z-series range.

Until the 80s, most mainframes used punched cards for input and teleprinters for output; these were later replaced by CRT (cathode ray tube) terminals. A typical (post 1980) 'mainframe' architecture is depicted in Figure 1.1. A terminal-based user interface would display screens controlled by the mainframe server using the 'virtual telecommunications access method' (VTAM)

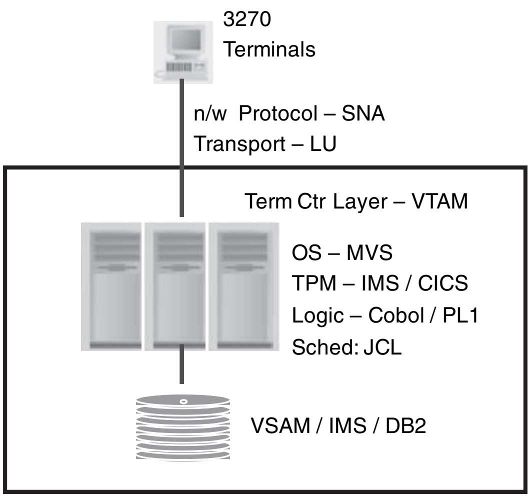

Figure 1.1. Mainframe architecture

for entering and viewing information. Terminals communicated with the mainframe using the 'systems network architecture' (SNA) protocol, instead of the ubiquitous TCP/IP protocol of today.

While these mainframe computers had limited CPU power by modern standards, their I/O bandwidth was (and is, to date) extremely generous relative to their CPU power. Consequently, mainframe applications were built using a batch architecture to minimize utilization of the CPU during data entry or retrieval. Thus, data would be written to disk as soon as it was captured and then processed by scheduled background programs, in sharp contrast to the complex business logic that gets executed during 'online' transactions on the web today. In fact, for many years, moving from a batch model to an online one was considered a major revolution in IT architecture, and large systems migration efforts were undertaken to achieve this; it is easy to see why: In a batch system, if one deposited money in a bank account it would usually not show up in the balance until the next day after the 'end of day' batch jobs had run! Further, if there was incorrect data entry, a number of corrective measures would have to be triggered, rather than the immediate data validations we are now so used to.

In the early mainframe architectures (through the mid/late 80s), application data was stored either in structured files, or in database systems based on the hierarchical or networked data model. Typical examples include the hierarchical IMS database from IBM, or the IDMS network database, managed now by Computer Associates. The relational (RDBMS) model was published and prototyped in the 70s and debuted commercially in the early 80s with IBM's SQL/DS on the VM/CMS operating system However, relational databases came into mainstream use only after the mid 80s with the advent of IBM's DB2 on the mainframe and Oracle's implementation for the emerging Unix platform. In Chapter 10 we shall see how some of the ideas from these early databases are now reemerging in new, non-relational, cloud database models.

The storage subsystem in mainframes, called 'virtual storage access mechanism' (VSAM), built in support for a variety of file access and indexing mechanisms as well as sharing of data between concurrent users using record level locking mechanisms. Early file-structure-based data storage, including networked and hierarchical databases, rarely included support for concurrency control beyond simple locking. The need for transaction control, i.e., maintaining consistency of a logical unit of work made up of multiple updates, led to the development of 'transaction-processing monitors' (TP-monitors), such as CICS (customer information control system). CICS leveraged facilities of the VSAM layer and implemented commit and roll back protocols to support atomic transactions in a multi-user environment. CICS is still in use in conjunction with DB2 relational databases on IBM z-series mainframes. At the same time, the need for speed continued to see the exploitation of so called 'direct access' methods where transaction control is left to application logic. An example is is the TPF system for the airline industry, which is still probably the fastest application-embedded TP-monitor around.

Mainframe systems also pioneered the large-scale use of *virtual machine* technology, which today forms the bedrock of cloud computing infrastructure. Mainframes running the VM family of 'hypervisors' (though the term was not used at the time; see Chapter 8) could run many independent 'guest' operating systems, such as MVS (popular through the 90s), to z-OS, and now even Linux. Further, virtual machine environments and the operating systems running on mainframes included high levels of automation, similar in many ways to those now being deployed in cloud environments, albeit at a much larger scale: Support for hardware fault tolerance included automatic migration of jobs if CPUs or memory units failed, as well as software fault tolerance, or 'recovery' facilities as pioneered in the MVS operating system. Fine-grained resource measurement, monitoring and error diagnostic capabilities were built into the mainframe architecture; such capabilities are once again becoming essential for cloud computing platforms.

Thus, we can see that far from being an academic exercise, during our excursion into mainframe history we have found many design features of the mainframe era that are now hallmarks of today's emerging cloud computing world; virtual machines, fault tolerance, non-relational databases, and last but not least, centralized computing itself. We now continue our historical tour beyond the mainframe era, continuing to look for early lessons that may stand us in good stead when we delve deeper into cloud computing architectures in later chapters.

1.3 CLIENT-SERVER ARCHITECTURE

The microprocessor revolution of the 80s brought PCs to business desktops as well as homes. At the same time minicomputers such as the VAX family and RISC-based systems running the Unix operating system and supporting the C programming language became available. It was now conceivable to move some data processing tasks away from expensive mainframes to exploit the seemingly powerful and inexpensive desktop CPUs. As an added benefit corporate data became available on the same desktop computers that were beginning to be used for word processing and spreadsheet applications using emerging PC-based office-productivity tools. In contrast terminals were difficult to use and typically found only in 'data processing rooms'. Moreover, relational databases, such as Oracle, became available on minicomputers, overtaking the relatively lukewarm adoption of DB2 in the mainframe world. Finally, networking using TCP/IP rapidly became a standard, meaning that networks of PCs and minicomputers could share data.

Corporate data processing rapidly moved to exploit these new technologies. Figure 1.2 shows the architecture of client-server systems. First, the 'forms' architecture for minicomputer-based data processing became popular. At first this architecture involved the use of terminals to access server-side logic in C, mirroring the mainframe architecture; later PC-based forms applications provided graphical 'GUIs' as opposed to the terminal-based character-oriented 'CUIs.' The GUI 'forms' model was the first 'client-server' architecture.

The 'forms' architecture evolved into the more general client-server architecture, wherein significant processing logic executes in a client application,

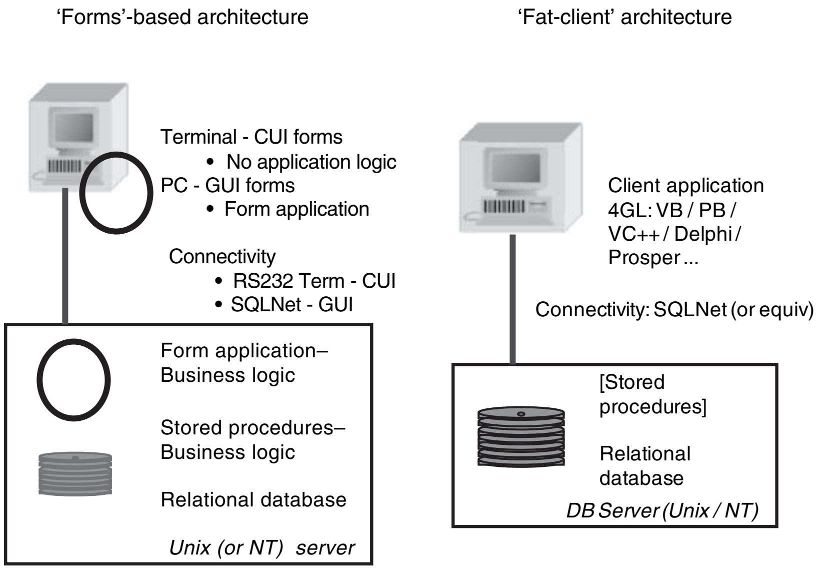

FIGURE 1.2. Client-server architectures

such as a desktop PC: Therefore the client-server architecture is also referred to as a 'fat-client' architecture, as shown in Figure 1.2. The client application (or 'fat-client') directly makes calls (using SQL) to the relational database using networking protocols such as SQL/Net, running over a local area (or even wide area) network using TCP/IP. Business logic largely resides within the client application code, though some business logic can also be implemented within the database for faster performance, using 'stored procedures.'

The client-server architecture became hugely popular: Mainframe applications which had been evolving for more than a decade were rapidly becoming difficult to maintain, and client-server provided a refreshing and seemingly cheaper alternative to recreating these applications for the new world of desktop computers and smaller Unix-based servers. Further, by leveraging the computing power on desktop computers to perform validations and other logic, 'online' systems became possible, a big step forward for a world used to batch processing. Lastly, graphical user interfaces allowed the development of extremely rich user interfaces, which added to the feeling of being 'redeemed' from the mainframe world.

In the early to mid 90s, the client-server revolution spawned and drove the success of a host of application software products, such as SAP-R/3, the client-server version of SAP's ERP software[2] for core manufacturing process automation; which was later extended to other areas of enterprise operations. Similarly supply chain management (SCM), such as from i2, and customer relationship management (CRM), such as from Seibel, also became popular. With these products, it was conceivable, in principle, to replace large parts of the functionality deployed on mainframes by client-server systems, at a fraction of the cost.

However, the client-server architecture soon began to exhibit its limitations as its usage grew beyond small workgroup applications to the core systems of large organizations: Since processing logic on the 'client' directly accessed the database layer, client-server applications usually made many requests to the server while processing a single screen. Each such request was relatively bulky as compared to the terminal-based model where only the input and final result of a computation were transmitted. In fact, CICS and IMS even today support 'changed-data only' modes of terminal images, where only those bytes

[2] SAP-R2 had been around on mainframes for over a decade.

changed by a user are transmitted over the network. Such ‘frugal’ network architectures enabled globally distributed terminals to connect to a central mainframe even though network bandwidths were far lower than they are today. Thus, while the client-server model worked fine over a local area network, it created problems when client-server systems began to be deployed on wide area networks connecting globally distributed offices. As a result, many organizations were forced to create regional data centers, each replicating the same enterprise application, albeit with local data. This structure itself led to inefficiencies in managing global software upgrades, not to mention the additional complications posed by having to upgrade the ‘client’ applications on each desktop machine as well.

Finally, it also became clear over time that application maintenance was far costlier when user interface and business logic code was intermixed, as almost always became the case in the ‘fat’ client-side applications. Lastly, and in the long run most importantly, the client-server model *did not scale*; organizations such as banks and stock exchanges where very high volume processing was the norm could not be supported by the client-server model. Thus, the mainframe remained the only means to achieve large throughput high-performance business processing.

The client-server era leaves us with many negative lessons: the perils of distributing processing and data, the complexity of managing upgrades across many instances and versions, and the importance of a scalable computing architecture. As we shall see in later chapters, many of these challenges continue to recur as wider adoption of the new cloud computing models are envisaged.

1.4 3-TIER ARCHITECTURES WITH TP MONITORS

Why did client-server architectures fail to scale for high volume transaction processing? Not because the CPUs were inferior to mainframes; in fact by the late 90s, RISC CPUs had exceeded mainframes in raw processing power. However, unlike the mainframe, client-server architectures had no virtual machine layer or job control systems to control access to limited resources such as CPU and disk. Thus, as depicted in Figure 1.3, 10 000 clients machines would end up consuming 10 000 processes, database connections, and a proportional amount of memory, open files and other resources, and thereby crash the server. (The numbers in the figure represent a late 90s view of computing, when 500MB of server memory was ‘too much,’ but the

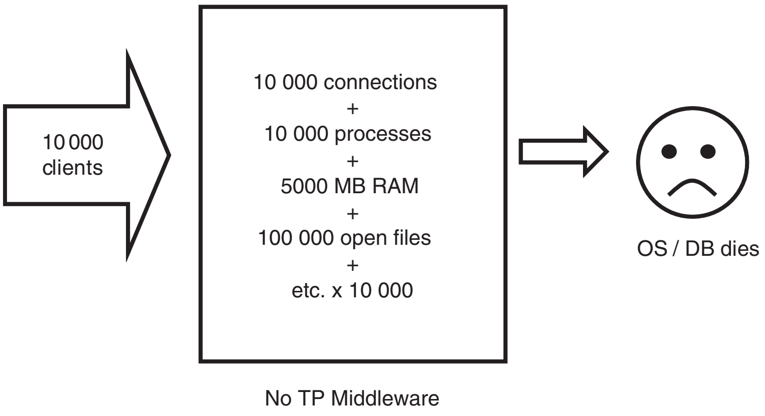

FIGURE 1.3. Client-server fails

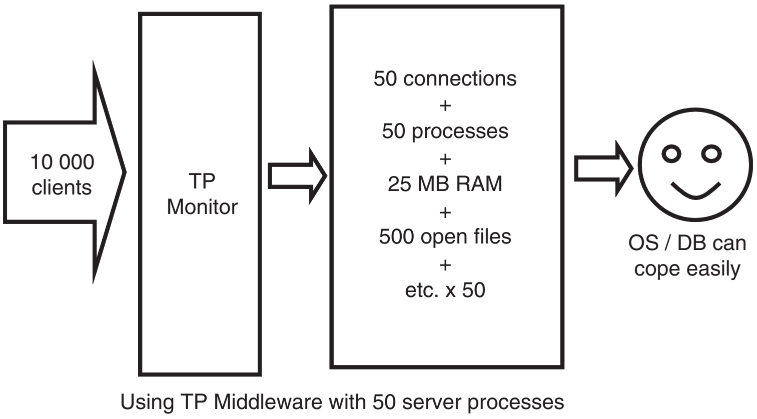

FIGURE 1.4. 3-tier architecture scales

principle remains the same even with the gigabytes of server memory available today.)

Transaction-processing monitors were *redeveloped* to solve this problem for midrange database servers. (Recall that the first TP monitors, such as CICS, were developed for mainframes.) These TP monitors were the first examples of 'middleware,' which sat between clients and a database server to manage access to scarce server resources, essentially by queuing client requests. Thus, as depicted in Figure 1.4, by limiting concurrent requests to a small number, say 50, the server could handle the large load while the clients only paid a small price in response time while their requests waited