Diseño y Evaluación de Infraestructuras Informáticas

Tema I

Análisis, monitorización y evaluación de rendimiento

Contenidos

- Rendimiento
- Ciclo de vida del sistema

I – Rendimiento

- Debemos diseñar/especificar/implementar sistema para producción
- Requerimientos del cliente:
  - Rendimiento de la aplicación, en producción, debe ser adecuado
  - Costes implementación deben ser adecuados
  - Costes explotación deben ser adecuados
- ¿Qué entiende cliente por "adecuado"?
  - Siempre terminaremos adoptando compromisos
  - *"The best is the enemy of the good enough"*
  - *"Good, Fast and Cheap – pick two"*

Rendimiento

- Concepto de "rendimiento" engloba diversos aspectos, y varía con cada tipo de aplicación

- Ejemplos de formas de evaluar el "rendimiento":
  - *Web interactiva* → Experiencia de usuario, tiempo carga páginas
  - *B.D. comercio electrónico* → Número transacciones/seg
  - Virtualización escritorio → Tiempo respuesta interactiva
  - Servidor ficheros en nube → Velocidad de transferencia

- Aplicación compleja puede involucrar varios criterios simultáneamente

Rendimiento

- Rendimiento puede valorarse por métricas internas del sistema:
  - ¿Cuánta carga tienen las CPUs?
  - ¿Cuántos procesos están activos? ¿Cuántos en espera?
  - ¿Cuánta memoria consumen los procesos en ejecución?
  - ¿Cuál es la carga de E/S del sistema? ¿Y sus latencias?
  - ¿Cuánto tráfico soporta la red del sistema?
  - ¿Qué cargas degradan inaceptablemente estas métricas?

Rendimiento

- Más ejemplos de métricas del sistema

System	Throughput Metric
OLTP System	Transactions per Second (tps)
	tpm-C [17]
Web Site	HTTP requests/sec
	Page Views per Second
	Bytes/sec
E-commerce Site	Web Interactions Per Second (WIPS) [18]
	Sessions per Second
	Searches per Second
Router	Packets per Second (PPS)
	MB transferred per Second
CPU	Millions of Instructions per Second (MIPS)
	Floating Point Operations per Second (FLOPS)
Disk	I/Os per Second
	KB transferred per Second
E-mail Server	Messages Sent Per Second

Rendimiento

- Usualmente, "rendimiento" significa medidas de satisfacción de usuario y respuesta del sistema

- ¿Encuentran los usuarios "lento" el sistema?
  - Si va lento, evitarán usarlo
  - Si va lento, clientes se irán a otro sitio web

- Métricas internas positivas son inútiles si experiencia global de usuario es negativa
  - Sistema responde tan lento como su módulo más lento
  - Si hay módulos lentos y rápidos mezclados, los módulos "buenos" están desaprovechados
    - Pagamos por rendimiento hardware desaprovechado

Rendimiento

- Ejemplos típicos de métrica respuesta sistema y experiencia usuario
  - Tiempo de respuesta
  - Tiempo en cargar una página dada
  - Tiempo en completar una transacción
  - Número de usuarios simultáneos, y tiempo de espera
  - Porcentaje de tiempo de disponibilidad del sistema
  - Límites de funcionalidades permitidas a usuario (p.e, número de archivos que puede almacenar)

Rendimiento

- Métricas pueden venir dadas por las especificaciones de diseño y funcionales del sistema
  - Ej: límite en las funcionalidades permitidas a los usuarios
- En otros casos, límites dados por arquitectura del sistema
- Métricas con impacto en explotación comercial suelen ir ligadas a SLAs
  - SLA = *Service Level Agreement*
  - Contrato que especifica por escrito límites mínimos aceptables de rendimiento
  - Incumplimiento suele ir ligado a penalización o compensaciones económicas

Rendimiento

- SLA puede dar sensación engañosa de confianza

Acme Hosting, Inc. will use commercially reasonable efforts to make the SuperHostingPlan available with a Monthly uptime percentage (defined below) of at least 99.9% during any monthly billing cycle. In the event Acme Hosting, Inc. does not meet this commitment, you will be eligible to receive a service credit as described below.

Monthly uptime percentage	Credit percentage
Between 99 and 99.9%	1 day credit
Less than 99%	1 week credit

Looks pretty reassuring, doesn't it? The problem is, 99.9% uptime stretched over a month isn't as great a number as one might think:

30 days = 720 hours = 43,200 minutes
99.9% of 43,200 minutes = 43,156.8 minutes
43,200 minutes – 43,156.8 minutes = 43.2 minutes

Rendimiento

- 99,999% = alta disponibilidad

TABLE 2-1. SLA percentages and acceptable downtimes

Uptime SLA	Downtime per year
90.0%	36 days, 12 hours
95.0%	18 days, 6 hours
99.0%	87 hours, 36 minutes
99.50%	43 hours, 48 minutes
99.90%	8 hours, 45 minutes, 36 seconds
99.99%	52 minutes, 33 seconds
99.999%	5 minutes, 15 seconds
99.9999%	32 seconds

Rendimiento

- Cuidado con aceptar cumplir SLAs muy severas
  - Pueden ser prohibitivamente caras de alcanzar

- Ej: montamos sistema a empresa hosting, y SLA especifica tiempo máximo de carga para cualquier página
  - En SLA sí habíamos incluido limitación a número usuarios concurrentes
  - Pero en SLA **no** habíamos limitado complejidad de las páginas
    - Resultan ser demasiado pesadas (imágenes, base datos, etc)
  - Cliente se niega a asumir costes ingeniería para mejorar diseño de páginas
  - Consecuencia: incumplimientos y penalizaciones

Rendimiento

- Moralejas:
  - Es imperativo hacer análisis de rendimiento y planificación de capacidad antes de asumir SLA
  - Deben considerarse todos los posibles factores involucrados
  - Mejor renunciar a un cliente que asumir SLAs peligrosamente arriesgadas
  - Aceptar cliente y SLA con un sistema inmaduro puede salir bien
    - Puede también salir horriblemente mal

II – Ciclo de vida del sistema

- Planificación capacidad del sistema será necesaria…
  - siempre, en la fase de diseño del sistema
  - posiblemente, durante la puesta en explotación del sistema

- Dependiendo del escenario, acertar con el cálculo inicial de la capacidad será:
  - importante
  - o crítico

Systems Development Life Cycle (SDLC)

Life-Cycle Phases

Diseño de crecimiento incremental

Initiation

Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created

System Concept Development

Defines the scope or boundary of the concepts. Includes Systems Boundary Document. Cost Benefit Analysis. Risk Management Plan and Feasibility Study.

Planning

Develops a Project Management Plan and other planning documents. Provides the basis for acquiring the resources needed to achieve a soulution.

¿Infraestructura?
¿Dimensionamiento?

Requirements Analysis

Analyses user needs and develops user requirements. Create a detailed Functional Requirements Document.

Design

Transforms detailed requirements into complete, detailed Systems Design Document Focuses on how to deliver the required functionality

¿Coste?
¿Presupuesto?
¡Decisiones!

Development

Converts a design into a complete information system Includes acquiring and installing systems environment; creating and testing databases preparing test case procedures; preparing test files, coding, compiling, refining programs; performing test readiness review and procurement activities.

Integration and Test

Demonstrates that developed system conforms to requirements as specified in the Functional Requirements Document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Evitamos "sorpresas" ya en casa del cliente

Implementation

Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the Integration and Test Phases

Operations & Maintenance

Describes tasks to operate and maintain information systems in a production environment. includes Post-Implementation and In-Process Reviews.

¿Todo OK?
¿Cumplimos las SLAs?

Disposition

Describes end-of-system activities, emphasis is given to proper preparation of data.

Ciclo de vida

- Procedimiento de ajuste iterativo de capacidad

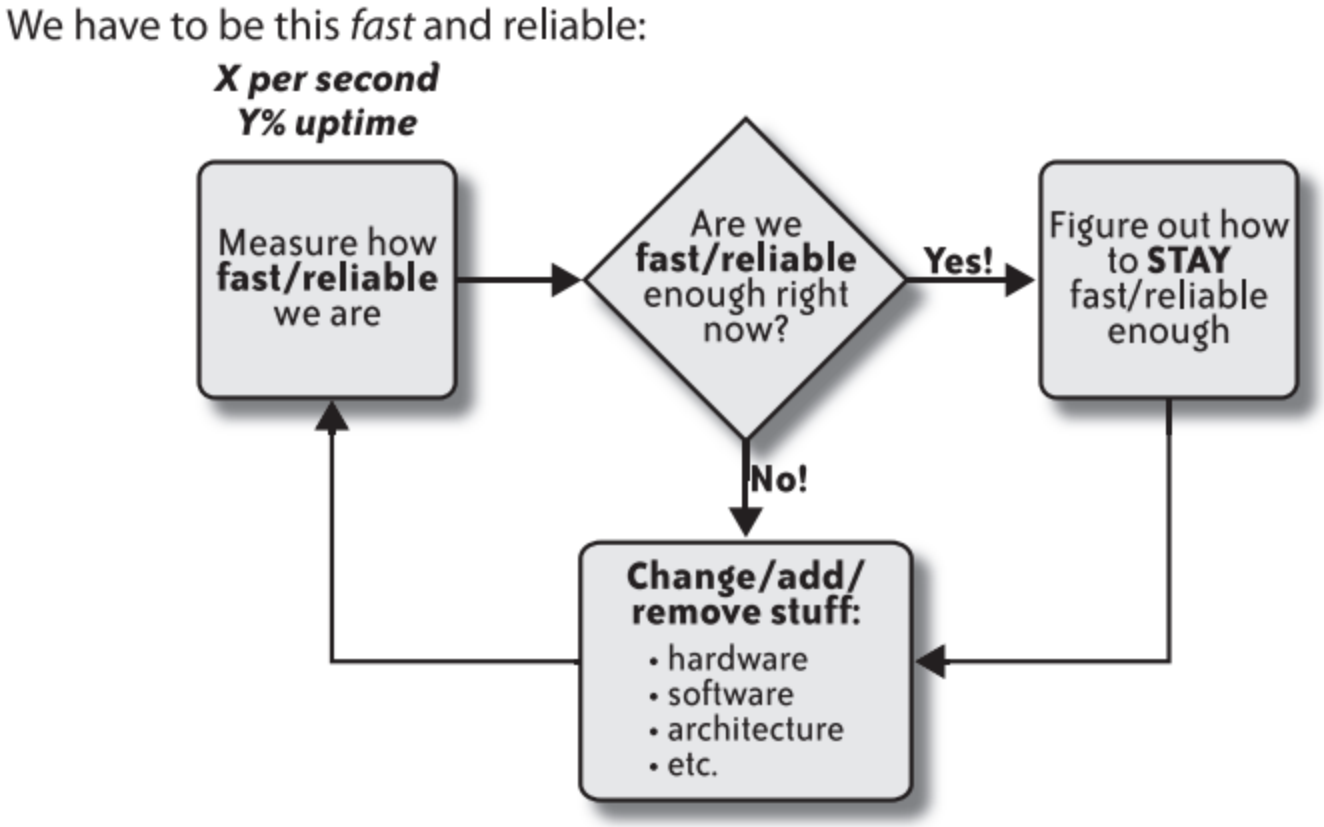


FIGURE 1-1. The process for determining the capacity you need

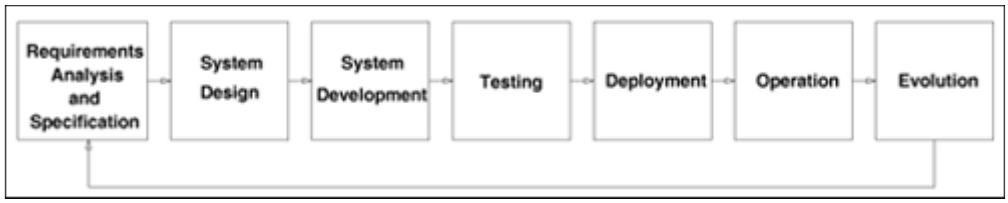
Performance by Design: Computer Capacity Planning by Example
Prentice Hall
ISBN-13: 978-0-13-090673-1

1.3. System Life Cycle

Addressing performance problems at the end of system development is a common industrial practice that can lead to using more expensive hardware than originally specified, time consuming performance-tuning procedures, and, in some extreme cases, to a complete system redesign [3]. It is therefore important to consider performance as an integral part of a computer system life cycle and not as an afterthought. The methods used to assure that that QoS requirements are met, once a system is developed, are part of the discipline called Performance Engineering (PE) [16].

This section discusses the seven phases of the life cyle of any IT system: requirements analysis and specification, design, development, testing, deployment, operation, and evolution as illustrated in Fig. 1.6. The inputs and outputs of each phase are discussed, the tasks involved in each phase are described, and QoS issues associated with each phase are addressed.

Figure 1.6. System life cycle.



1.3.1. Requirements Analysis and Specification

During this phase of the life cycle of a computer system, the analysts, in conjunction with users, gather information about what they want the system to do. The result of this analysis is a requirements specifications document that is divided into two main parts:

- *Functional requirements:* The functional requirements specify the set of functions the system must provide with the corresponding inputs and outputs as well as the interaction patterns between the system and the outside world (users). For example, the functional requirements of an online bookstore could indicate that the site must provide a search function that allows users to search for books based on keywords, ISBN, title, and authors. The specification indicates how the results of a search are displayed back to the user. The functional requirements usually include information about the physical environment and technology to be used to design and implement the system. In the same example, the specification could say that the online bookstore site should use Web servers based on UNIX and Apache and that it should also provide access to wireless users using the Wireless Application Protocol (WAP) [19].

- *Non-functional requirements:* The non-functional requirements deal mainly with the QoS requirements expected from the system. Issues such as performance, availability, reliability, and security are specified as part of the non-functional requirements. A qualitative and quantitative characterization of the workload must be given so that the QoS requirements can be specified for specific workload types and levels. For example, a non-functional requirement could specify that "at peak periods, the online bookstore is expected to receive 50 search requests/sec and respond within 2 seconds to 95% of the requests."

1.3.2. System Design

System design is the stage in which the question "How will the requirements be met?" is answered. In this phase, the system architecture is designed, the system is broken down into components, major data structures, including files and databases, are designed, algorithms are selected and/or designed, and pseudo code for the major system components is written. It is also during this phase that the interfaces between the various components are specified. These interfaces may be of different types, including local procedure calls, Remote Procedure Calls (RPC), and message exchanges of various types.

The current trend in software engineering is to reuse as many proven software solutions as possible. While this approach is very attractive from the point of view of shortening the duration of the design and development phases, it may pose risks in terms of performance. Designs that perform well in one type of environment and under a certain type of workload may perform very poorly in other settings. For example, a search engine used in a low volume online retailer may perform very poorly when used in an e-commerce site that receives millions of requests per day. As the workload intensity scales up, different techniques, different algorithms, and different designs may have to be adopted to satisfy the non-functional requirements.

A key recommendation is that special care be given to the non-functional requirements at the design stage since decisions made at this stage are more likely to have a strong impact on system performance, availability, reliability, and security. Moreover, problems caused by poor decisions made at this stage are much more expensive and time consuming to correct than those generated by decisions made at the later stages of the development life cycle.

It is also common at the design stage to make decisions related to the adoption of third-party components such as messaging middleware, search engines, directory services, and transaction processing software. Again, it is important to evaluate the performance impact of each of the third-party solutions on overall system performance. Credible performance evaluation is a non-trivial task

1.3.3. System Development

During this phase, the various components of the system are implemented. Some may be completely new creations, others may be adapted from existing similar components, and others may just be reused without modification from other system implementations. Components are then interconnected to form the system. As there are many possible ways to design a system that meets the requirements, there are also many different implementation decisions, left open at the design stage, that can significantly affect performance. For example, it may be left to the development phase to decide how a particular search to a database will be implemented. The developer must not only make sure that the query returns the correct answer but also that its performance will be acceptable when the query is submitted to a production database with potentially millions of records as opposed to a small test database.

As components are developed, they should be instrumented to facilitate data gathering for the testing phase and for the QoS monitoring that takes place during system operation. It should be easy to selectively turn on and off the instrumentation code of components to avoid unnecessary overhead generated by data collection.

1.3.4. System Testing

System testing usually occurs concurrently with system development. As components become available, they can be tested in isolation. This is called unit testing. Then, tested components are put together into subsystems which are further tested until the entire system meets its specification requirements.

It is common for a significant amount of effort to be invested in testing the functional requirements while not enough resources are devoted to the testing of the non-functional requirements such as performance, scalability, availability, and security. When performance is tested before deployment, the usual approach is to conduct *load testing* [10, 12]. In this case, scripts of typical transactions are constructed and executed on the system while its performance is measured. These scripts can simulate an increasing number of users, called *virtual users*.

While testing is an important part of a computer system life cycle, it is not possible to anticipate or test all possible scenarios because of time and budget constraints. Therefore, virtually every moderate to complex system is deployed without being fully tested for both functional and non-functional requirements. To reduce the chance that flaws go unnoticed, one must use design and development techniques that attempt to build correct, reliable, secure, and well-performing systems from the ground up.

1.3.5. System Deployment

After a system has been tested, usually in a controlled environment, it is deployed for use. During system deployment, many configuration parameters (e.g., maximum number of TCP connections, maximum number of threads, timeout periods, database connection pool size) have to be set for optimal performance.

1.3.6. System Operation

A system in operation has to be constantly monitored to check if the QoS requirements are being met. Examples of features that should be monitored include:

- *Workload:* determination of peak periods during which the system is subject to higher workload intensity levels, determination of the characteristics of the arrival process of requests (e.g., does the workload exhibit extreme bursts?), and detection of unusual patterns that could indicate security attacks such as Denial of Service (DoS) attacks. Part of the workload monitoring process includes a characterization of the global workload into “similar” types of

requests. This is important since the performance of the system depends on the types of requests it receives.

- *External Performance Metrics:* measurement of user-perceived satisfaction and statistics (e.g., mean, standard deviation, percentiles) relative to response time, throughput, and probability that requests are rejected. When establishing monitoring procedures it is important to keep in mind that for some applications (e.g., Web-based applications), the response time perceived by a user depends not only on the system—the Web site in that case—but also on the user's geographical location, bandwidth of the Internet connection, the time of day, and on the local machine performance characteristics.
- *Internal Performance Metrics:* identification of internal factors that aid in the diagnosis of performance failures and bottleneck detection. Examples include the utilization of processors, storage devices, and networks, and the number of requests waiting in the various software and hardware queues. The amount of information collected this way can easily become overwhelming. Care must be taken to efficiently and effectively organize, collect, and report such internal performance metrics. There are several monitoring and performance management tools that provide good filtering, visualization, and alarm-based reporting capabilities. Some of the tools use data-mining techniques to find useful correlations between internal and external metrics.
- *Availability:* determination of the percentage of time that a system is available to service requests. This is usually done by external monitoring agents that send requests to a system at regular intervals to determine if the system is responsive. Availability determination may be done at various levels. Consider for example an online bookstore that has several Web servers and a load balancer that distributes incoming HTTP requests to the Web servers. The load balancer may periodically send "heart-beat" pings to each server to check its network connectivity. In addition to this, there may be software agents running at computers spread over several geographical regions that send search requests to the online bookstore at regular intervals. The latter type of monitoring is useful to check the availability of the service as a whole, including the entire site and the networking infrastructure that connects users to the site. It is important that such pings are infrequent enough so as not to interfere with the normal workload, but are frequent enough to provide accurate information in order for corrective action to be taken in a timely fashion.

During system operation, it may be necessary to change the values of the various configuration parameters to adapt to the evolving nature of the system workload so that the QoS requirements are continuously met.

Ciclo de vida

- "Ciclo de vida" del sistema = aproximación sistemática a diseño, implementación y explotación del sistema

- Metodología diseño y planificación capacidad asociada dependen de naturaleza de aplicación
  - No se puede usar misma metodología dimensionamiento para diseños de crecimiento incremental que para diseños finalistas

Ciclo de vida

- Escenario 1: Diseño de sistema con crecimiento incremental (aplicación web)
  - Ej: start-up ofrece a público servicios basados en web
  - Aplicación dinámica = cambios de funcionalidad frecuentes
  - Disponibilidad inicial de capital limitada
  - Éxito de start-up no está garantizado
  - Diseño inicial debe ser compromiso entre rendimiento y coste
    - Si tiene éxito, la capacidad se irá ampliando incrementalmente

Ciclo de vida

- Adecuación capacidad requerirá análisis iterativo y continuo

- Revisión incremental capacidad sistema debe considerar:
  - ¿Cuántas visitas tiene sitio web?
  - ¿Cuántas sesiones/usuarios se crean por día/hora?
  - ¿Cuánto tráfico de red se genera?
  - ¿Se ha degradado el rendimiento del sistema?
  - ¿Cuáles son las proyecciones de crecimiento?
  - ¿Hay disponibilidad de nuevo capital?

Ciclo de vida

- Si se observan caídas de rendimiento, hay dos opciones:
  - Mejorar comportamiento hardware (inversión en infraestructura)
  - Mejorar comportamiento software (inversión en ingeniería)

- Mejoras posibles de rendimiento hardware (escalado):
  - Mejor hardware (misma arquitectura, servidores más potentes)
    - Aumentar RAM, CPU, discos
    - *Escalado vertical* (*Scale-up*, *vertical scaling*)
  - Más hardware (más máquinas, distribuir funcionalidad y carga)
    - Se vuelve a usar el mismo tipo de servidor ya existente
    - *Escalado horizontal* (*Scale-out*, *horizontal scaling*)

Ciclo de vida

- Mejora comportamiento software…
  - Implica identificar y eliminar cuellos de botella
  - Cuidado con costes de ingeniería asociados
    - Puede ser más rentable simplemente comprar más hardware
  - Recordar: diseño debe ser compromiso entre coste (total) y funcionalidad

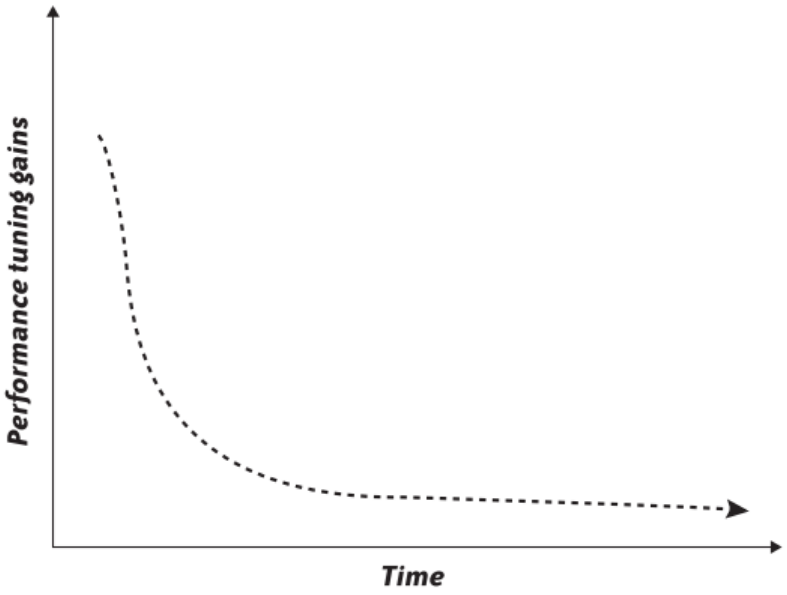


FIGURE 1-3. Decreasing returns from performance tuning

Ciclo de vida

- Escenario 2: Diseño de sistema con arquitectura "finalista"
  - Ej: archivo multimedia para uso interno en una institución
- Disponibilidad presupuestaria importante, pero única en el tiempo
  - Ej: concurso público, o inversión puntual de una empresa
- No es posible crecimiento incremental
  - Diseño estático: una vez puesto en marcha no se modificará en años
- Dimensionamiento debe ser correcto desde el principio
- Se desea minimizar el coste sin comprometer rendimiento
  - Dimensionamiento debe considerar peor caso posible
  - Debe considerar también necesidades de futuro próximo

Ciclo de vida

- Dimensionamiento para peor caso implica que para operación habitual estará sobredimensionado
  - Si el coste es excesivo, será necesario compromiso entre coste y posible degradación del rendimiento
  - Si cliente requiere este compromiso, debe ponerse por escrito
    - No siempre es posible, por especificaciones de concursos
  - ¿Perder concurso, o aceptar requerimientos difíciles de cumplir?
    - Decisión difícil
    - Requiere conocer bien límites sistema y respuesta del hardware
    - A veces, lo mejor es decir “no”