

Towards an Efficient Reliable and Low Cost Integrated Information System in The Organization

S. G. FOUNTOUKIS and G. E. HARAMIS

Abstract—Herein, the difficulties concerning the procurement of an organization's integrated information system - IIS that might arise during the determination procedure of the system configuration, are investigated. These difficulties are attempted to be overcome by classifying the many alternative technical options offered, together with recommendations according to the needs of the prospective information system owners. Modern technologies, (e.g. object oriented databases) are examined along with their corresponding traditional alternatives and are also recommended, according to their suitability.

The recommendations are not based in sole technical features but financial considerations are also taken into account. Low cost solutions based on reliable open source software and similar products or valid and economic machine architectures are suggested.

The IIS should not be viewed as a set of discriminate parts, which have interesting technical features, but as a functioning integrated set that serves the needs of the external and internal clients of the organization. It must be consisting of high performance software running in fast and efficient hardware infrastructure. The capability of the whole system to serve multiple client requests concurrently and efficiently, rejecting none or the least of them if necessary, along with the minimization of the financial cost, should be the key factors in the determination process. Combining different Mathematical methods that can model both the performance of the system and the client request arrivals, the number of processors or nodes of the server systems of the IIS, can be estimated. The tasks of the organization's human group that has undertaken the responsibility to determine the correct configuration of the IIS, for procurement purposes, are to employ these methods for the determination of the IIS configuration, along with their efforts to keep the budgeted within the given limits.

Index Terms – high performance software, high performance architectures, performance estimation, fraction of rejected requests.

I. INTRODUCTION

THE role of the computer technology, in an organization is to support an easy information flow throughout all its departments. Computerizing all aspects of the organization processes is essential for an optimal resource use. A certain configuration of a set of both software and hardware parts, consists of an integrated information system, which can support the entire organization functionalities. The performance of the system is critical, affecting the flow of the information. A poor flow can make difficult the picture forming of the organization as a whole and creates the risk of declination of the resource use rate.

Previous works [1], [2] having to do with the determination of the configuration of computer systems, do not take into account the many technical and financial options offered in both the machine architectures and the suitable software, for efficient, reliable and low cost systems. In the field of the estimation of the computer systems performance and consumption [3], [4], [5], interesting works exist and efforts have been spent [6], [7] for relating the system performance with hardware selection and design parameters. But, none of these, combines results derived from computer systems performance models for determining the configuration of an organization's information system, for procurement purposes.

S. G. Fountoukis is with the Dept. of Informatics with Applications in Biomedicine, University of Central Greece, 2-4 Papasiopoulou Str, Lamia 35100, Hellas, (e-mail: sfount@phs.uoa.gr).

G. E. Haramis is with the Department of Business Administration, University of Macedonia, 156 Egnatia Str, Thessaloniki 54006, Hellas, (e-mail: haramis@uom.gr)

Herein, for the most of the basic software infrastructure of the system, options between commercial and freely available or open source software technologies are suggested, in each case considering their advantages and / or disadvantages. If special software packages are needed to be developed, requirements and restrictions about the software architectures, development methodologies and tools are specified towards efficient, reliable and easily maintained software. Also, for the organization's hardware infrastructure, suitable and low cost machine architectures for servers and workstations are recommended. Mathematical methods that can model the performance of the computer systems are used to assist the determination of their configuration.

The procurement of an efficient and low cost computerization infrastructure requires special attention and must be regarded as a project, according to the rule that every activity in an organization is a project. In this case the financial cost must be taken into account and the technical and the functional requirements of the desired infrastructure must be determined.

II. THE HARDWARE CHOICE

From the hardware infrastructure point of view, an information system in an organization must be composed by servers and workstations. Concerning the servers there are several options: symmetrical multiprocessors - SMPs, clusters and massively parallel processing - MPP each having its advantages and disadvantages according to the purpose they will serve. Intimate connections are established amongst the processors of the SMPs, so they are known as *tightly coupled machines*. They are consisting of multiple processors, residing in one cabinet, managed by the same operating system, connected via a high-bandwidth link, sharing the same memory and having equal access to I/O devices. Clusters are sets of interconnected systems, each having its own processor, memory, I/O and storage devices. Each system is called node and works using the same copy of an operating system. The connection between the nodes of a cluster usually is established via fast local area network topologies. The cluster should be viewed as a *single system* by the users, hiding the fact that it is a set of systems. This is not always easily achievable, but efforts are spend towards this end. It must also be *homogenous*, that is all the nodes must be similar, running the same copy of the operating system. The size of a cluster is of order ten, and all of the nodes must be physically placed *near each other* in a limited space. MPPs are also sets of systems, like the clusters, however the number of nodes is not limited to several tens, but to hundreds or even thousands. They are interconnected using specialized high performance network topologies.

It is possible to combine two server architectures. A cluster of SMPs is a combination of SMP machines connected through a fast network topology to build a cluster.

Grid computing is related to cluster computing; however it is distinguished by considerable differences. A grid is a, via the internet, commonly established connection of heterogeneous sets of systems, which can be dispersed in great distances. The sets are not each other completely trusted since they belong in different organizations. Grids can be utilized for special computing purposes, but the related applications can be run during prescheduled time intervals only.

Workstations in the past were uniprocessor systems. In the present time most common laptop and desktop systems are in fact entry level SMPs, since dual - core processor computing systems dominate the markets. The recent introduction of quad - core processors, predicts a near future where the workstations will be able to run high performance applications utilizing four or even more processing units. Such availability of hardware systems ensures the ability of running multi - programming and multi - processing applications in both servers and workstations.

For the determination of the hardware infrastructure of the information system in an enterprise, choices must be made, regarding the servers and workstations, amongst the several options described above.

If the primary purposes of a server system are the high availability, the maintenance easiness and the compatibility with the workstations, then the best choice can be a *failover* cluster. This system, utilizing its natural abundance (and independence) of nodes, which provide it with high intrinsic availability, can continue working with the remaining nodes in case of a node failure. It has an advantage over a SMP

system, which needs restarting in order to function with fewer processors in the corresponding situation where a processor fails. The independence of nodes in a cluster system also facilitates its maintenance. While a node is out of order, the necessary updates can be installed and the maintenance duties can be performed. When everything is completed, the node can be integrated again in the cluster. Since a cluster is a set of independent computer systems, which can be similar to workstations, the required compatibility is ensured. A minimal cluster ensuring availability can be consisted of only two nodes.

The web based information systems and the database systems of an enterprise require high availability. The latter are the most common software systems tuned for clusters, therefore they suit in a cluster environment. Applications related to decision support or to online transaction processing in financial, industrial and other organizations, requiring intense data entry and retrieval, are favored by database applications running in cluster systems. Therefore, cluster systems are recommended in cases where high availability of database systems is required.

If a web based information system of an enterprise, accepts concurrently many calls through the internet (or intranet), then a *load - balancing* cluster system is a good choice, able to handle the heavy input work. This type of cluster accepts and passes all the input workload through a small set of front - end nodes, which distributes it amongst the rest of the nodes, called back - ends. The load - balancing cluster, known as *server farm*, also has high availability features.

The MPPs are also suitable systems in cases where cluster systems are required, provided that a large number of nodes is needed.

Other types of software applications (except databases discussed previously), especially commercial applications, need serious modifications to become able to utilize the parallelism of the multiple nodes of a cluster. Besides, there are not well established standards for writing parallel programs for cluster systems and the software tools for this purpose are limited. Also the communications between processes running on different nodes are not very efficient, since they are based on message passing through the network than on shared memory (as in case of SMPs). Therefore special care must be applied to minimize node interaction during applications development, which is destined to run on clusters. Under these circumstances, the necessary software development (or modification), requires serious research and development efforts even for small software packets, which can be proved a process of increased time and financial cost for the organization. Therefore, the use of high performance computing clusters is limited to research and scientific purposes only. In spite of these facts, the necessary hardware of a cluster infrastructure is very common and has a low economic cost. This is an additional reason to be highly recommended as a server system in an organization, in cases where both the handling of heavy input workload and the high availability of databases and/or web based information systems, are required.

If multithreaded commercial, financial or similar applications, needing inter thread communication, must be run and accessed concurrently by a large number of workstations, in an organization, then a SMP server system is a necessary choice. The inter - process and inter - thread communications of multi - programming and multi - processing applications running on the multiple processors of a SMP machine, are easily realized through the shared memory of the system. However, the SMPs have their disadvantages requiring attention. Any software or hardware error in a SMP machine can cause a failure of the operating system, which in its turn can cause a collapse to the complete system. Also, the whole SMP system must be stopped for the necessary maintenance activities to take place. The financial cost of an SMP system generally is higher compared to a corresponding cluster system. It must be noted, that the architecture of the system and the type of the processors used for its implementation, considerably affect its cost. If the implementation is based on *Risc* (non *x86*) architecture processors, then the cost is usually increased compared to on common *x86* architecture based implementation cost. Since the driven by the market current trend in hardware technology is the wide use of *x86* architecture dual core and quad core processors, which are suitable for workstations, this cost can be reduced if the SMP is implemented with this type of processors.

If both the high performance of multithreaded applications and the availability of the system are required, then the suitable server system is a cluster of SMPs.

If the organization needs to run special applications, over extended time periods (e.g. simulations or stochastic models), then the use of Grid computing can be a recommended choice, since it is a low cost solution. Grid can be used for computing jobs, which have to be independent and do not require inter-process communication. Grid allocates packets of workload to the nodes, which work independently of each other and can store their results in a shared storage.

The storage system is an essential subsystem of a server. It deserves special care, since the information (and thus the data) should be continuously available through both the internet and intranet. Half of the cost of a server system is allocated to storage subsystem. The choice for the suitable storage subsystem is dependent upon the types of the applications that run on a server.

If the applications require intense data entry and retrieval (e.g. database transactions) or the management of huge amount of data, which can grow even more with the pass of time (e.g. decision support systems), then the recommended choice is a *storage area network* – SAN subsystem, which is a fast local network, usually built using fibre channels, that connects storage devices. It uses commands for data access that correspond to low level (e.g. SCSI) disk commands. Data transfer is achieved through block I/O (low level).

If the application running in a server is content management oriented, then a *network attached system* - NAS is indicated as the proper storage subsystem. This is due to the facts that the storage devices are distributed across a network and the data are accessed applying file system commands, whereas the latter are used to manipulate content. That is, data transfer is achieved through file I/O (high level). The usual file systems utilized by this storage system are the *network file system* - NFS and the *common internet file system* - CIFS. Therefore data transfers are easily achieved applying high level commands.

Other options for data storage include iSCSI and DAS systems. The first distributes the storage devices across the internet and uses an implementation of the SCSI protocol over IP for data access. It is an internet extended SAN system. The second refers to *directly attached storage*, indicating that the storage devices are directly attached to the machine, using any protocol (it can be SCSI). This system utilizes the file system of the installed operating system. It is the simplest of all storage systems and is common to all workstations.

The SAN, iSCSI and NAS systems allow data sharing at the level of files. The first two systems, should transform the data access from block oriented to file oriented using a NFS or CIFS metadata server. The NAS system directly supports data sharing due to its nature.

For the organization users that do not run other types of applications (e.g. spreadsheets, word processors, etc), except the applications running on a SMP server system, which should be computationally powerful enough to handle the concurrent requirements by the users, the workstations can be replaced by simple terminals, avoiding the unnecessary cost. For all other users, fully functional workstations can be provided.

III. THE SYSTEM SOFTWARE CHOICE

The system software should support the hardware workings. System software includes operating systems for both servers and workstations and the software supporting the storage systems. Usually the vendor of the storage system provides the necessary software, supporting the functionalities of its product and there is no need for compatible software to be provided from other vendors. Therefore the determination of the type, the characteristics and the vendor of the storage system also determines its necessary software.

The operating system controls the resources of the machine, the processes and the threads of the applications, the input and output devices, it manages the file systems and memory etc.

The Unix-like operating systems can run on many computer system architectures. Linux and FreeBSD are Unix-like systems and they can run on common x86 platforms. They are very popular and they can be installed on servers and also on workstations. SUN Solaris, a Unix system, can run on both Risk and x86 system architectures. HP-UX and IBM's AIX are also Unix systems and are designed to run only on hardware provided by their vendors. Linux and SUN Solaris have a great availability of software utilities and device drivers. However, Unix systems provided by various vendors have a considerable financial cost.

The Solaris operating system is an exception, since it is offered free. Linux and FreeBSD systems also are offered free (open source systems). All the Unix based operating systems have good features and outstanding stability.

Microsoft Windows are widely used on desktop and laptop systems, but the installation of this operating system on a large number of workstations can have a considerable cost for an organization due to licensing. Windows is an operating system that performs well on low-end and mid-range machines and it can also be used on clusters.

For the servers, the free offered Unix based operating systems are recommended for the financial and technical reasons discussed above. For the workstations there are two options; either the cost of Windows licensing must be accepted, or a minimal personnel training in the Linux environment must be undertaken. If the personnel adaptation to the Linux environment will be proved successful, the latter choice can financially benefit the organization in the long run.

IV. THE APPLICATION SOFTWARE CHOICE

The application software of an organization can be of many types. Enterprise resource planning, decision support systems, customer relationship management, financial, supply chain management, e-learning, internet based interactive applications are only few examples of application software types.

Database management systems - DBMS are necessary subsystems of applications and functionally support them. Currently two models of database systems dominate the market; the relational - RDBMS and the object oriented - OODBMS. The development of the majority of applications is based on relational databases. However advanced applications successfully employ object oriented databases to handle huge amounts of data. It must be noted that object oriented application software is directly compatible with the object databases, whereas the compatibility with the relational databases is achieved through special software bridges. The latter can have a negative impact on the system performance.

If a high performance processing application, with high concurrency requirements, is needed to process huge amounts of data in complex form, then the utilization of an object oriented database for data storage and retrieval purposes is advised. Applications related to financial, telecommunication, transportation, medical, government, defence and scientific areas can run successfully in organizations that employ object databases in their back end systems.

The largest database system in the world, at the Stanford Linear Accelerator Center, holding successfully, at least, 1000 terabytes of data, is based on object oriented technologies.

Amongst the object databases, commercial and open source products exist.

Ozone is an open source OODBMS implemented in Java, suitable for many types of applications, which is multi-threaded and can serve multiple users concurrently.

FrameD is also an open source specialized object database suitable for knowledge base applications.

Versant is a commercial object database system, which is also multithreaded and has many other nice features.

For the relational databases, also commercial and open source implementations exist.

Depending on the type of the server, there are several relational database options.

For a cluster that shares the storage system, the Oracle real application cluster - RAC is a product that permits the Oracle database to be installed on the cluster nodes. For clusters that do not share the storage (share nothing), IBM's DB2 for Windows and Unix-like environments and also Microsoft's SQL Server for Windows are the alternatives.

For SMP machines, all the well known databases are tuned to work on them; SQL Server, Oracle, DB2 etc. However, the above relational database systems are commercial and have considerable financial cost. Alternatives to commercial relational databases are the open source systems; MySQL and Postgress are the most common of them.

For share nothing clusters, MySQL cluster is a database system that supports automatic node recoverability and ensures that an entire system can be safely recovered in a consistent state even in extreme cases of

failure of all the nodes. It also has a good performance, since it is a main memory clustered database, which keeps all data in memory and limits I/O bottlenecks by asynchronously writing transaction logs to disks. Since, it is a free offered database system, having competitive features, it must be considered as a choice for a database system of a low cost, efficient and reliable information system.

For the application software itself, there are also two options; either an off-the-shelf commercial software must be customized or a new version must be built from “scratch”. The decision is dependent on the software complexity. If the customization requires heavy efforts, may a better choice be the building of a new version. In this case, certain development methodologies, architectural and design patterns must be employed to ensure that the new version of the software will be efficient, reliable and its maintenance will be easy.

The object oriented technology is widely used today, therefore the *unified software development process* - USDP is the most suitable methodology to be followed for the software development. This methodology is also known as unified process or Rational unified process - RUP, due to the fact that it was created by the Rational Software Corporation, a division of IBM. It is an *iterative* software development process framework, which uses the unified modeling language - UML to create abstract models of the software systems under development. UML attempts to specify and visualize software systems. For this purpose, it employs both static and dynamic diagrams. UML can also be used for the representation of organizational structures, business process modeling and systems engineering modeling. Therefore it is the most suitable modeling language to describe the organization's needs in terms of models, which also will be the software models for development. Due to its iterative character the unified process can accomplish a more complete software development, compared to development methodology which is based on the traditional *waterfall* model. The latter is a sequential - linear software development model, without iterations, which passes through the sequence of its phases, flowing steadily towards the end of them. In this model, there is no return to previous phases to correct or complete them, whereas the extensions and the corrections of previous phases are the essence of the iterative character of the unified process.

Effective application software development requires the use of both architectural and design patterns to provide the system with the desired fundamental structural organization and to avoid complexities that may arise during the implementation process. Applying design structures that frequently are used to solve problems in similar situations, at the subsystem level, the complexities can be avoided.

The most common software architecture that is used in applications over a network is the *Client - Server* architecture. It is also known as the *two - tier* architecture. An application, running in a server and having a certain network address, can accept and respond to calls from other applications running in workstations, across the network, called clients.

The *three - tier* architecture expands the Client - Server architecture adding a new module. Each module runs in a separate platform and a change to a module does not affect the others. In this architecture, the server consists of two parts; the application server and the database server. The servers are linearly connected. A request arriving from a client must be processed first by the application running in the application server. Then it is directed to the database running in the database server. A response from the client follows the reverse way. Adding new modules linearly connected and running them in new separate platforms the system can be expanded to *n - tier* architecture.

If the high performance and the easiness of the maintenance are the dominant factors in a web interactive application, then the *model, view, controller* - MVC concept is the recommended architectural pattern. It separates the code that supports the functionality of the system (model - business logic) from the user interface (view) and uses a module that controls the data movement (controller). The system forms a triangle; it is not linear like the *n - tier* architecture. The *n - tier* and the MVC architectures can be combined, since they solve different architectural problems. The advantage of the MVC architecture is obvious. The user interface, which is usually a set of web pages, is free of other code, except html elements; therefore it can be easily loaded to the browsers of the clients. Besides, a change made in a module (e.g. in the functionality of the system - business logic), does not affect the rest of the system.

If issues related to communications between objects, at the subsystem level, must be faced during the application software development, then the application of patterns that belong to *behavioral* group of design patterns, can resolve the complexities.

For problems related to object creation, the *creational* design patterns can solve them, since they use controlled procedures to create objects.

If the problems that may arise are related to relationships between independent classes of objects, then the design can be easier if *structural* design patterns are applied.

For multi - threaded applications or applications with many processes, the *concurrency* patterns are suitable to resolve complex issues.

The *fundamental* patterns can be applied in all types of applications if fundamental software issues are to be faced.

V. MODELING THE CLIENT REQUEST ARRIVALS

Continuous time Markov chains - CTMC are suitable tools for the modelling of the arrivals of the client requests in servers.

Let $t \in [0, +\infty)$ be a continuous variable, $S = \{0, 1, 2, \dots\}$ be a countable set called set of states and $X(t)$ be a variable, which denotes the state at time t . $X(t)$ can change states while the time variable t runs.

A stochastic process $\{X(t) : t \geq 0\}$ is a continuous time Markov chain iff $P(X(t+s) = j | X(t) = i, \{X(z) : 0 \leq z < t\}) = P(X(t+s) = j | X(t) = i) = p_{ij}(s)$, $\forall t, s \in [0, +\infty)$, $\forall i, j \in S$. That is, the Markov chain has the memoryless property.

The probability $p_{ij}(s)$ represents the transition probability between the states i and j that needs time s to occur. The s is also called transition or intermediate time and is often denoted by T_i . During the time T_i , it holds $X(t) = i$.

The intermediate times T_i , $i \in S$ are exponentially distributed. Thus, $\forall i \in S$ there exists a constant $q_i > 0$ called rate, such that $P(T_i > t) = F(t) = e^{-q_i t}$ and $P(T_i = t) = f(t) = q_i e^{-q_i t}$, $t \geq 0$. The exponential distribution has the memoryless property, that is $\forall i \in S$ it holds $P(T_i > t+s | T_i > s) = P(T_i > t)$, $\forall t, s \in [0, +\infty)$. It also holds $E(T_i) = 1/q_i$, $\forall i \in S$. If $X(t) = i$, the rate q_i is interpreted as the rate out of state i .

If $X_n = X(t_n)$, denotes the state which occurs at time t_n , then $\{X_n\}$ represents a discrete time Markov chain with transition probabilities $p_{ij} = P(X_{n+1} = j | X_n = i)$.

Therefore, the continuous time Markov chain can be seen as the transition matrix $P = (p_{ij})$, consisting of the above discrete time Markov chain transition probabilities, accompanied by the set $Q = \{q_i : i \in S\}$, which is consisting of the rates of the intermediate times T_i , $i \in S$.

The state $j \in S$ is reachable from state $i \in S$ iff $\exists t \geq 0 : P(X(t) = j | X(0) = i) = P_{ij}(t) > 0$. Two states $i, j \in S$ communicate iff i is reachable from state j and vice versa. A continuous time Markov chain is called irreducible iff all its states communicate. Also the chain is irreducible iff its corresponding discrete chain is irreducible.

For a state $i \in S$ that is visited, the C_{ii} represents the continuous time until i is revisited. The state $i \in S$ is recurrent iff $P(C_{ii} < \infty) = 1$. Otherwise the state i is transient. Also, for the continuous time chain the state $i \in S$ is recurrent or transient iff for the corresponding discrete time chain the state i is recurrent or transient, respectively.

Let $i \in S$ be a recurrent state for a continuous time Markov chain. The state $i \in S$ is positive recurrent iff $E(C_{ii}) < \infty$, i.e. the expected return time is finite, and $i \in S$ is null recurrent iff $E(C_{ii}) = \infty$, i.e. the expected amount of time to return is infinite. A state $i \in S$ can be positive recurrent for the continuous time chain and null recurrent for the corresponding discrete time chain. The reverse is also true. In an irreducible continuous time chain, the states either are all positive recurrent or all null recurrent or all transient.

For the function $X(t)$, $t \geq 0$, with $X(t) \in S$ and for a state $j \in S$, the number of times that the $X(t)$ enters the j equals the number of times that it leaves the j , in the long run ($t \rightarrow \infty$), since for a departure from state j , a previous arrival into state j must be occurred. In other words, $\forall j \in S$ the rate into j equals the rate out of j .

An irreducible continuous time Markov chain is positive recurrent iff the set of balance equations: $a_j p_j = \sum_{i \neq j} p_i a_i p_{ij}$, $j \in S$, has a unique probability solution: $p_j \geq 0$, $j \in S$ with $\sum_{j \in S} p_j = 1$. That is, an irreducible continuous time Markov chain is positive recurrent iff $\forall j \in S$ the rate into j equals the rate out of j . A continuous time Markov chain that has a finite state space is positive recurrent. In this case, the balance equations have the unique probability solution.

A. Modelling M/M/1 Single Queue Server

Client requests arrive to a single service thread through a single FIFO queue. This type of arrivals follows the Poisson distribution at rate λ . Therefore it holds $P(U(t) = k) = e^{-\lambda t} \frac{(\lambda t)^k}{k!}$, $k = 0, 1, 2, \dots$, where $U(t)$

represents the number of arrivals that have occurred in the interval $[0, t)$.

The interarrival times U are independent and follow the exponential distribution at rate λ , independent of the Poisson process. Thus $P(U > t) = e^{-\lambda t}$, $t \geq 0$.

The service times S_n of the client requests are also independent and follow the exponential distribution at rate μ , independent of the Poisson process. Thus $P(S_n > t) = e^{-\mu t}$, $t \geq 0$.

Let $X(t)$ represents the number of client requests in the system at time t . The variable $X(t)$ can change its state, only if a new request arrives or if the request that is in service is completed. It must be noted that if $X(t) = n$, $n \geq 1$, then $n-1$ requests are waiting in the FIFO and 1 request is ongoing (in service). In this case, if a new request arrives at time $t_a > t$ then $X(t_a) = n+1$ or if the request in service is completed at time $t_c > t$ then $X(t_c) = n-1$.

The set Q of rates can be calculated as follows:

If $X(t) = 0$ then only a request can arrive next. Therefore, the intermediate time T_0 until the next arrival, is in fact an interarrival time U which follows the exponential distribution at rate λ . It holds: $q_0 = \lambda$.

For $X(t) = i \geq 1$, the intermediate times are given by $T_i = \min\{S_r, U\}$ where S_r is the remaining service time of the customer in service, and U is the time until the next arrival - interarrival time (both S_n and U have the memoryless property). It holds $P(T_i > t) = P(\min\{S_r, U\} > t) = P(S_r > t, U > t) = P(S_r > t) P(U > t) = e^{-\mu t} e^{-\lambda t} = e^{-(\mu+\lambda)t}$. Thus, $q_i = \mu + \lambda$, $i \geq 1$.

The discrete time Markov chain transition probabilities p_{ij} can be calculated as follows:

If $X_n = 0$, then the request in service is just completed, while no other request is in the queue and only a new request can arrive next. Thus $P(X_{n+1} = 1 | X_n = 0) = 1$, which implies $p_{0,1} = 1$.

For $X_n = i \geq 1$, if an arrival occurs first then $p_{ij} = P(U < S_r)$ else the request in service will be completed first and $p_{ij} = P(S_r < U)$.

Since $P(U < S_r | U = t) = P(S_r > t | U = t)$ and the S_r, U are independent, it follows $P(U < S_r | U = t) = P(S_r > t) = e^{-\mu t}$. The U is exponentially distributed at rate λ , that is $f(t) = \lambda e^{-\lambda t}$. Therefore $P(U < S_r) = \int_0^{\infty} P(U < S_r | U = t) \lambda e^{-\lambda t} dt = \int_0^{\infty} e^{-\mu t} \lambda e^{-\lambda t} dt = \lambda \int_0^{\infty} e^{-(\mu+\lambda)t} dt = \frac{-\lambda}{\mu + \lambda} \int_0^{\infty} e^{-(\mu+\lambda)t} d\{-(\mu + \lambda)t\} = \frac{\lambda}{\mu + \lambda}$. Due to $P(U$

$> S_r) + P(U < S_r) = 1$, it follows that $P(U > S_r) = \frac{\mu}{\mu + \lambda}$.

The first M in the notation $M/M/1$ means that the client requests arrive following the Poisson distribution (memoryless), the second M means that the service times follow the exponential distribution (also memoryless) and the 1 means that there is only one thread for servicing the requests. In the following notation $M/M/k$ the k means that the number of threads is k .

B. Modelling M/M/k Single Queue Server

Client requests arrive to a multithread server through a single FIFO queue. The server threads can serve k requests concurrently. The arriving requests enter the free available service threads through a queue. If $n \geq k$

requests have entered the system, then k request are in service and n-k are in the queue. The arrival of requests, the interarrival times and the service times are as in the M/M/1 case.

The minimum S_r amongst the service times S_{r_i} , $i=1,2,\dots,m$, $m \leq k$ determines the next transition of the system due to service completion: $P(S_r > t) = P(\min\{S_{r_i}, i=1,2,\dots,m, m \leq k\} > t) = P(S_{r_1} > t, S_{r_2} > t, \dots, S_{r_m} > t) = P(S_{r_1} > t)P(S_{r_2} > t) \dots P(S_{r_m} > t) = e^{-\mu t} e^{-\mu t} \dots e^{-\mu t} = e^{-m\mu t}$, $m \leq k$. As in the case M/M/1, the intermediate times are given by $T_i = \min\{S_r, U\}$ where U is the time until the next arrival. Therefore, T_i is exponentially distributed at rate $\lambda + m\mu$, $m \leq k$.

For $X(t) = i$, if $i < k$ then S_r and T_i are exponentially distributed at rates $i\mu$ and $\lambda + i\mu$, respectively, else $i \geq k$ and the corresponding rates become $k\mu$ and $\lambda + k\mu$.

For the discrete time Markov chain, it holds $p_{0,1} = 1$ and $p_{i,i+1} = \lambda/(\lambda + i\mu)$, $p_{i,i-1} = i\mu/(\lambda + i\mu)$, for $0 < i < k$, or $p_{i,i+1} = \lambda/(\lambda + k\mu)$, $p_{i,i-1} = k\mu/(\lambda + k\mu)$, for $i \geq k$.

C. Modelling M/M/k No Queue Server

Client requests arrive to a multithread server without queue. Any request that finds all the k service threads busy is rejected.

Let $X(t)$ represents the number of requests that are in service by the corresponding threads at time t. For $\lambda, \mu > 0$ the continuous time Markov chain of the system is irreducible with finite state space $S = \{0, 1, 2, \dots, k\}$. Therefore, it is positive recurrent. The balance equations are:

$$\lambda p_0 = \mu p_1$$

$$(\lambda + \mu) p_1 = \lambda p_0 + 2 \mu p_2$$

$$(\lambda + 2 \mu) p_2 = \lambda p_1 + 3 \mu p_3$$

...

$$(\lambda + j \mu) p_j = \lambda p_{j-1} + (j+1) \mu p_{j+1}, \quad 0 \leq j \leq k-1.$$

Applying the first equation into the second and the second into the third and so on, it concludes:

$$\lambda p_0 = \mu p_1$$

$$\lambda p_1 = 2 \mu p_2$$

$$\lambda p_2 = 3 \mu p_3$$

...

$$\lambda p_j = (j+1) \mu p_{j+1}, \quad 0 \leq j \leq k-1,$$

From the last equation it is derived:

$$p_{j+1} = p_j \frac{\rho}{j+1}, \quad \text{where } \rho = \lambda/\mu, \quad 0 \leq j \leq k-1.$$

Solving these equations recursively the following result is obtained: $p_j = p_0 \frac{\rho^j}{j!}$, $0 \leq j \leq k$.

Summing these equations to one: $1 = \sum_{j=0}^k p_j = p_0 \sum_{j=0}^k \frac{\rho^j}{j!}$, yields $1 = p_0 (1 + \sum_{j=1}^k \frac{\rho^j}{j!}) \Rightarrow p_0 = (1 + \sum_{j=1}^k \frac{\rho^j}{j!})^{-1}$.

Therefore from $p_j = p_0 \frac{\rho^j}{j!}$, $0 \leq j \leq k$, it results: $p_j = (1 + \sum_{n=1}^k \frac{\rho^n}{n!})^{-1} \frac{\rho^j}{j!}$, $0 \leq j \leq k$. Applying it for $j = k$ the final

result is obtained: $p_k = (1 + \sum_{n=1}^k \frac{\rho^n}{n!})^{-1} \frac{\rho^k}{k!}$.

This is the fraction of time that all the service threads are busy. This is also the fraction of rejected requests, that is, the fraction of client request arrivals who find all the k service threads busy.

Indeed, the client request arrivals follow Poisson distribution at rate λ . Let π_j represents the fraction of client request arrivals, which find j service threads busy, t_n represents the time of the n^{th} arrival and $X(t)$ represents the number of busy service threads at time t. It is defined:

$$\pi_j = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N 1\{X(t_n) = j\}, \text{ where } 1\{X(t_n) = j\} \text{ equals } 1 \text{ if } X(t_n) = j \text{ and } 0 \text{ otherwise.}$$

$X(t)$ makes transitions from state j to state $j+1$ at rate $\lambda\pi_j$, since these transition can only occur when arrivals find j server threads busy. But from the balance equations above: $\lambda p_j = (j + 1) \mu p_{j+1}$, $0 \leq j \leq k-1$, it results that λp_j also expresses the same rate.

Therefore $\lambda\pi_j = \lambda p_j \Rightarrow \pi_j = p_j$, $0 \leq j \leq k$. Thus the fraction of client request arrivals which find j service threads busy is equal to the fraction of time where the j threads are busy.

VI. MODELING THE SERVER SYSTEM PERFORMANCE

Operational analysis methods can be used to model the server systems.

These systems can be classified in three types. The open system receives and processes requests and sends back the results. The results received by the clients do not generate new requests. There is no session support in this system. In the closed system, the results of the request processing are sent back to the clients and they produce new requests to the server. The closed system with delay station is a more realistic closed system. After the reception of the results, the client waits for a time (called think time) and then sends again a new request to the server.

The operational research methods utilize values that are directly measurable in a single experiment within a finite time, using a prototype system.

A server system is consisting of many partial resources denoted by integer values $i \neq 0$. However the whole system can also be considered as a resource and it is denoted by 0 ($i=0$) or no number at all. For each resource i , a set of measures should be taken during a finite time T of observation. The measures are values for the quantities that model the system performance. These quantities are:

- 1) Arrival rate $\lambda_i = A_i/T$, where A_i is the number of arrived client requests.
- 2) Rate of completions or average throughput $\mu_i = C_i/T$, where C_i is the number of the serviced client requests.
- 3) Average service time $S_i = B_i/C_i$, where B_i is the total service time and C_i is the number of the serviced client requests.
- 4) Utilization $U_i = B_i/T$, where B_i is the total service time which also is the total busy time. It can also be written as $U_i = \mu_i S_i$, since $\mu_i S_i = (C_i/T)(B_i/C_i) = B_i/T = U_i$.
- 5) $V_i = C_i/C_0$, the fraction of the visits to the resource i . C_i is the number of completions (serviced client requests) for the resource i , whereas C_0 is the number of completions for the whole system. Since $\mu_i = C_i/T \Rightarrow C_i = \mu_i T$, the fraction can also be written as $V_i = \mu_i / \mu_0$.
- 6) The rate of completions or average throughput can also be written as $\mu_i = \mu_0 V_i$, since $V_i = \mu_i / \mu_0$.
- 7) Total service demand $D_i = CR_i V_i$, where CR_i is the number of client requests and V_i is the fraction of the visits to the resource i .
- 8) $Q_i = \mu_i R_i$, is the average number of client requests that are in service, for the resource i of a closed system, where R_i is the response time of the resource i . $Q_0 = \mu_0 R_0$ applies for the whole closed system. If the number of the resources in a closed system is n , it holds $Q_0 = \mu_1 R_1 + \mu_2 R_2 + \dots + \mu_n R_n$, which implies $Q_0/\mu_0 = (\mu_1/\mu_0)R_1 + (\mu_2/\mu_0)R_2 + \dots + (\mu_n/\mu_0)R_n$ and gives $R_0 = V_1 R_1 + V_2 R_2 + \dots + V_n R_n = \sum_{i=1}^n V_i R_i$. For an open system as a whole, the average number of client requests can be written as $N_0 = \mu_0 R_0$. N_0 is the total number of client requests in the whole system.
- 9) Interactive systems based on client - server architectures are typical closed system with delay stations. The users of these systems, having received server responses, wait for a time before they send their new requests. If the users wait for time T_w and the server responds after R time, then $T/(T_w + R)$ is the number of requests of a user, within time T . It is supposed that all the user requests are serviced. The total number of requests, of N_u users is $T N_u / (T_w + R)$. So, the system throughput is $\mu = T N_u / (T_w + R) T \Rightarrow \mu = N_u / (T_w + R)$, which implies $R = (N_u / \mu) - T_w$. This is the interactive response time, denoted by $R_{int} = R$.

- 10) The busiest resource of a system has the highest service demand D_{\max} . This can be identified by $D_b = D_{\max} = \max_i \{D_i\}$, where b indicates the busiest resource. Potential bottlenecks can be detected utilizing the D_b .
- 11) For a closed system with delay stations, the throughput and the response are given by: $\mu(Nu) \leq \min \left\{ \frac{1}{D_{\max}}, \frac{Nu}{D+Tw} \right\}$, and $R(Nu) \geq \max \{D_{\max}, NuD - Tw\}$.
- 12) For an open system, the throughput is given by: $\mu(Nu) \leq \frac{1}{D_{\max}}$, i.e. only the busiest resource limits the system.

In the second part of the relation (see 11) concerning the throughput for closed systems, two linear equations are defined. The diagram of the first is a parallel line to the horizontal axis and the diagram of the second is a sloping line. These lines intersect in a point. The same happens with the second part of the relation concerning the response time. The two points of the two intersections of the two pairs of diagram lines have the same horizontal axis projection. That is, the two intersections correspond to the same number of clients (client requests that are served by threads). This number of clients corresponds to a system that its performance begins to saturate. It is a critical point that places a limit to the number of client requests that can be served by a system, at a satisfactory level of performance. This number is denoted by N_{crt} .

To obtain the N_{crt} , the two pairs of linear equations must be solved separately.

For the throughput, the equations are $\mu(Nu) = \frac{1}{D_{\max}}$ and $\mu(Nu) = \frac{Nu}{D+Tw}$. Solving $\frac{1}{D_{\max}} = \frac{Nu}{D+Tw}$, the

required result $N_{\text{crt}} = \frac{D+Tw}{D_{\max}}$ is obtained. For the response time, the pair of equations gives exactly the same result.

Therefore $N_{\text{crt}} = \frac{D+Tw}{D_{\max}}$, is the number of service threads that can run in a processor or node before the server system saturates.

VII. METHODOLOGY FOR ESTIMATING THE NUMBER OF PROCESSORS OR NODES OF A SERVER

Measurements should be taken from a similar system running the organization's software application or from a simple prototype system. These measurements will be utilized in the modeling of both the client request arrivals, employing stochastic methods, and the server system under determination, employing operational analysis.

For the estimation of the maximum number of the service threads that will be run concurrently in the whole server system, servicing the client request arrivals, a satisfactory upper bound - UB must be placed to the

fraction of the rejected requests (ch. V): $(1 + \sum_{n=1}^k \frac{\rho^n}{n!})^{-1} \frac{\rho^k}{k!} \leq \text{UB}$. Then the number k must be found

(numerical methods can be used) that satisfies this equation.

From the maximum number k of the service threads that will be run concurrently in the whole server system, the number of the processors or the nodes can be estimated. This can be achieved through the knowledge of the maximum number of threads per processor or node.

Modelling a simple server system with a single processor or node, the number of service threads that can satisfactory run in the processor or node, is given (ch. VI) by the formula: $N_{\text{crt}} = \frac{D+Tw}{D_{\max}}$. Therefore the

number of processors or nodes of the whole server system under determination is $F = \frac{k}{N_{\text{crt}}}$. Applying the

operational analysis methods, the resources of the system that can constitute bottlenecks can be detected. In these cases, the technical features of the potential bottlenecked resources should be revised and new features must be determined for the bottlenecks to be removed.

In client - server architectures with heavy workload, where only a (relatively small) predefined fixed number of clients (i.e. monitors or workstations) are serviced by a server system, the number of the service threads should be equal to the number of clients. In this case, the estimation of the number of service threads is equivalent to the estimation of the number of the workstations or monitors and vice versa.

After the determination of the configuration of the system, the simple prototype system used for measurement purposes can be scaled up to meet the requirements of the determined system.

VIII. CONCLUDING REMARKS

The procurement of an efficient, reliable and low cost integrated information system – IIS of an organization, needs the determination of its configuration. This determination can be assisted by the use of a classification of the many technical options offered, together with suggestions regarding both technical and financial considerations. Amongst the many existing combinations for reliable solutions, low cost options based on open source technologies and economic and efficient machine architectures are recommended. Also, the use of the combination of different mathematical modeling methods, to determine both the number of processors or nodes of the servers and the number of the workstations of the information system, facilitates the configuration determination procedure. To achieve this purpose, the number of concurrent client requests in service is estimated and emphasis is placed in the minimization of the fraction of the rejected client requests.

REFERENCES

- [1] Hua-Yang Lin, Ping-Yu Hsu, Gwo-Ji Sheen, (2007), *A fuzzy-based decision-making procedure for data warehouse system selection*, Expert Systems with Applications: An International Journal, v.32 n.3, pp.939-953.
- [2] E. M. Timmreck, (1973), *Computer Selection Methodology*, ACM Computing Surveys - CSUR, Volume 5, Issue 4, pp. 200 – 222.
- [3] Raj Jain, (1991), *The Art of Computer Systems Performance Analysis*, John Wiley & Sons.
- [4] Daniel A Menasce, Virgilio A F Almeida, (2002), *Capacity Planning for Web Services – Metrics, Models and Methods*, Prentice Hall
- [5] Niv Ahituv, Magid Igbaria, (1988), *A model for predicting and evaluating computer resource consumption*, Communications of the ACM, v.31 n.12, pp.1467-1473.
- [6] A.-W. Scheer, (1977), *Combination of an optimization model for hardware selection with data determination methods*, ACM SIGSIM Simulation Digest, v.8 n.4, pp. 51-62.
- [7] S. Rannem, V. C. Hamacher, S. G. Zaky, P. Connolly, (1974), *On relating small computer performance to design parameters*, ACM SIGARCH Computer Architecture News, v.3 n.4, pp.146-151.