1987

Distributed Systems Based Upon a Functional Communication Model with a Timed Token Protocol

Dan C. Marinescu

Report Number:
87-669
DISTRIBUTED SYSTEMS BASED UPON
A FUNCTIONAL COMMUNICATION MODEL
WITH A TIMED TOKEN PROTOCOL

Dan C. Marinescu

CSD-TR-669
March 1987
DISTRIBUTED SYSTEMS BASED UPON A FUNCTIONAL COMMUNICATION
MODEL WITH A TIMED TOKEN PROTOCOL

Dan C. Marinescu
Department of Computer Science
Purdue University
West Lafayette, IN 47907

Abstract

The paper investigates communication in distributed systems with a non-homogeneous architecture. Such systems consist of a large number of user nodes (workstations) and a smaller number of server nodes (computational servers, file servers, printer servers, etc.). In the framework of the functional communication model, we propose a token passing scheme which allows both server initiated communication and communication initiated by the users. A timed token protocol is used to control the two access modes, the scheduled access and the nonscheduled one.

1. OVERVIEW OF FUNCTIONAL COMMUNICATION

Communication plays a central role in the design of any distributed system. Common communication models are centered around the concept of moving data from one node of a network to another. An alternative communication model based upon the concept of moving computations from one node to another in a network, is possible. In the following, we call the later a functional communication model. Its chief idea is that communication occurs only in connection with a request for service.

The duality of the two modes seems obvious; the Remote Procedure Call protocols are commonly implemented using the data movement paradigm. On the other hand, data can be moved in a functional communication model, either by regarding it as a result of a request for service or as input data from/to the remote server. The motivation for a functional communication model is determined by the following question:

Since the client-server paradigm is a cornerstone in the design of distributed systems, why not use an underlaying communication model which can support it in a simple, direct and efficient way?

The answer to this question should be based on a qualitative as well as a quantitative analysis.

Unfortunately, the quantitative performance analysis is by no means trivial for any distributed
system of reasonable complexity, hence we'll review only some qualitative arguments against and in favor of the new model.

First of all, the functional communication model which is less general than the traditional one, might prove to be good for distributed systems built around a broadcast channel, but it is unclear whether it would be acceptable for an interconnection of such systems. Second, it requires sophisticated network interfaces capable to process multiple fields of an incoming control data packet. For example, in case of ADMA systems to be described in the next section, the interface has to examine a function selection field of an incoming control packet to determine whether it is allowed to send a service request for a particular service, queued at that node. If the comparison is successful, the interface has to determine whether the length of the data packet, which contains the service request is smaller than the residual data count in the control packet. Only when both conditions are satisfied, the interface is allowed to send the request. In addition to increased complexity, the network interface must carry out these functions at a high speed in order to ensure a low station latency.

Another significant aspect to be mentioned is that, in order to have an efficient implementation of the functional communication model, the communication channel must assure a high transmission speed in the Gbps range. The maximum packet size on such channels can be increased by one to two orders of magnitude, as compared with existing LANs to allow data packets of the order of $10^6$ bits. Then, a large percentage of service requests, as well as results of computations, would fit into a single data packet, and functional communication could be implemented efficiently.

The last two limitations will most likely be overcome due to technological advances in computer technology, which will allow the design of sophisticated interfaces, and in the communication technology which already provides very high speed fiber optics communication channels.
Let us now examine the positive side of functional communication. First of all, it leads to a simpler communication architecture for a distributed system built around a broadcast channel. The communication paradigm is self-contained, it does not need to introduce artificial means for flow control and error control as the common paradigm does. The very fact that a client receives a semantically correct answer to a service request, eliminates the need for acknowledgements for the individual data packets containing the request. Since the client blocks waiting for the request to be processed, the need for flow control is also eliminated.

There are even more significant aspects of this simplicity. It is highly desirable to achieve location independence of any service. Clearly, this leads to increased flexibility and system reliability. Functional communication achieves this goal in a very elegant manner.

On the other hand, a client process is interested in having a service request processed in the shortest possible time and with the minimum effort to locate the server which can provide the service subject to these constraints. For this reason, connection-oriented communication has given way to connectionless communication in many distributed systems. But still in case of connectionless communication, the sender has to bind this request for service to a particular server by specifying the address of the server before sending any data packet. If multiple servers can provide the same service, the client process has to select one of them based upon status information. Any significant change in the status of the server occurring since the last reported status information, like the server is no longer available, could make the choice of the client nonoptimal or even undesirable.

In our model, we introduce the functional connection between a client process and a functional group, the group of server processes capable to provide the service. The functional connection allows the latest possible binding between a client and a server. The client needs to know only that the service is provided by some server in the distributed system, and it generates a request stamped only with the service id. A server in the service group will bind to it provided
that its current status allows it.

Clearly this does not guarantee the shortest possible response time for the client process, since it is possible that after a server has accepted the request for processing, another server down the line might be the optimal one. Nevertheless, the model does simplify the server selection process performed by a client, and in the same time, opens the possibility of efficient load sharing algorithms among the servers.

In fact, each server executes a threshold scheduling, and accepts a service request depending upon the threshold. Such a scheme is considerably simpler than bidding schemes and can be comparable to them in terms of load balancing properties if the threshold is properly chosen.

The functional model is not restrictive, since a client process can still select a server by specifying in addition to a service id, the address of desired server process, whenever the need occurs. Several mechanisms to allow communication among clients can be built into the model, the most obvious one being the use of a dedicated communication server. In addition to client to client communication, this server will support functions related to internetworking.

A more detailed discussion of functional communication can be found in [6]. One final remark is that the distributed system being designed today, consists of a large number of workstations connected with computational servers, printing servers, file servers, etc. The functional communication model matches well this type of distributed system architecture.

In conclusion, functional communication seems to be an iconoclastic approach to distributed system design. While layering is emphasized by traditional communication models, we propose to integrate low level functions usually found at the MAC (Medium Access Control) layer of a local network with high level functions like load sharing in a distributed system.

The arguments in favor of the functional communication model are its simplicity, increased functionality, and probably, better performance.
2. SCHEDULED AND NONSCHEDULED ACCESS IN A FUNCTIONAL SYSTEM

The distributed systems examined are non-homogeneous; the nodes belong to two different functional classes, server and users, and heavier traffic is expected to be associated with server nodes. The systems are asymmetric, since each server provides a different assortment of services. Another aspect of asymmetry is related to the input and output traffic in a server node.

To accommodate such systems we can use a functional communication model with communication initiated either by the clients, as discussed in the previous section or with communication initiated by the servers as in case of ADMA (Availability Driven Multiple Access) systems. The first approach has several limitations, namely, there is no guarantee that any server in the service group will accept the service request, and it is also possible that the number of servers accepting a request will be larger than the number of parallel executions the client needs.

To eliminate these limitations we have introduced ADMA schemes in which servers have priority over the users in controlling the communication channel [8]. Essentially, in an ADMA system the servers execute a multiple access algorithm, which eventually selects a server, to get the control of the channel. Then the server advertises the set of services it is willing to provide and based upon this information, a group of users is selected. The group consists of the users which need the services advertised by the server in control. The group executes a multiple access algorithm, which eventually allows each member of the group to send their service requests. A \(<\text{token passing, token passing}>\) scheme is described in reference [8] and a \(<\text{token passing, collision resolution}>\) is described in [7], where \(<x,y>\) is a notation with the following significance: \(x\) is the multiple access algorithm among servers and \(y\) the one among users.

The \(<\text{token passing, y}>\) class of ADMA systems seems the most interesting one, but a cycle time analysis proves to be difficult. Such systems are in fact multi-polling systems.

The delay analysis necessary to determine the response time is hard. Hence our main concern, namely that the response time may increase in an ADMA system since the sending of a
request is delayed until the station is polled, could not be answered in the general case.

One could easily imagine a starvation scenario in which user station $i$ which requests service type $j$ cannot send such a request simply because server $k$ which can provide this service, is heavily loaded and it is unwilling to accept any new requests for a few CAP cycles. A CAP is a control packet acting like a token in the ring of the servers. Then, when server $k$ starts accepting again, it becomes overloaded again before station $i$ manages to send its request, and so on.

An alternative scheme to eliminate this client starvation effect, is described in the following. The scheme combines ADMA with client initiated communication. The scheme works as before, in the sense that the control over the channel is passed from one server to another by means of a token. But when a server has control, one recognizes two epochs. The “scheduled access” corresponds to the epoch when the server actively exercises his control over the channel. First it sends data packets (containing eventually the results of previous processing) then polls user stations and accepts the service requests sent by them, as in an ADMA system. But, provided that conditions permit, a second epoch is started this epoch corresponds to a “non-scheduled access”. During this epoch, the server sends a different type of polling. A user station receiving this polling message can transmit freely any service request, and not only the ones advertised by the server in control. The two epochs are controlled by means of a timed token protocol.

The server acceptance policy is based upon a two level threshold algorithm. Each server, say, $S_i$ establishes two thresholds, $t^l_i$ and $t^h_i$. They will be called the low water mark and respectively the high water mark. If the load of server $i$, $L_i$ is below $t^l_i$, then the server advertises its service when it receives CAP, and obviously accepts them. If $t^l_i < L_i \leq t^h_i$ the server does not advertise its services, but accepts an unscheduled request sent by a user in need. When $L_i > t^h_i$, the server does not accept any request.
3. ACCESS CONTROL VIA A TIMED TOKEN PROTOCOL

The concept of a timed token was introduced by Grow and by Ulm [11]. Based upon this idea, a protocol for transmission rates in the range of 100 Mbps, the FDDI (Fiber Distributed Data Interface) Token Ring Protocol was proposed and is currently in the standardization phase. The FDDI protocol is designed to allow stations to transmit two types of traffic: a synchronous traffic, which has delivery time constraints, and an asynchronous traffic, which has no time constraints.

To satisfy the traffic requirements, each station has an allocated bandwidth for synchronous traffic and can use it whenever the token visits the station. To determine whether the station can transmit the asynchronous traffic, a target token rotation time (TTRT) is chosen and made known to all stations in the network. Each station monitors the token cycle time, defined as the time elapsed since the token was last passed by the station to its successor, and the time the token arrives again to the station. If the measured cycle time is less than the target cycle time, it means that the token has arrived early and the station can transmit some of its asynchronous traffic after it uses its allocated quota for the synchronous traffic. Sevcik and Johnson have analyzed the cycle time properties of a timed token system [9], and we’ll use the notation and the results obtained by them in a slightly different framework.

If $T$ is the target cycle time and $C_{c,i-1}$ is the actual token cycle time as observed by station $i$, during the $c$-th visit of the token, then the timed token protocol operates under the following conditions:

(C1) The synchronous transmission of any station $i$ is upper bounded by its allocated quota:

$$g_{c,i} \leq f_i = f_i \cdot T$$

The sum of allocated quotas for all stations should not exceed the total channel bandwidth:
\[ \sum_{i=1}^{n} f_i \leq 1, \] (3.2)

when \( n \) is the total number of stations and \( f_i \) is the fraction of the total channel bandwidth allocated to station \( i \).

(C2) The asynchronous transmission allowed for station \( i \) is upper bounded by the earliness of the token arrival:

\[ a_{c,i} \leq \max[0, T - C_{c,i-1}] \] (3.3)

with \( C_{c,i} \) defined as:

\[ C_{c,i} = \sum_{j,k = c-1,i+1}^{c,i} (g_{j,k} + a_{j,k}) \] (3.4)

(C3) No transmission (asynchronous or synchronous) is allowed during the first cycle and no asynchronous transmission is allowed during the second cycle.

The protocol operating under these conditions has the following properties:

P1 - The average token cycle time \( E(C_{x,y}) \) is upper bounded by the target token cycle time

\[ E(C_{x,y}) \leq T \] (3.5)

P2 - The maximum token cycle time is upper bounded by twice the target cycle time

\[ \max(C_{x,y}) \leq 2T \] (3.6)

4. PROTOCOL DESCRIPTION

In the following we consider a token passing ring in which the token is embedded in a control packet which carries out additional control functions. Similar systems have been proposed by Andrews and Schultz [1]. In our case there are two logical rings collapsed in a physical ring. There are \( N \) stations in the ring of users and \( M \) stations in the ring of servers. The following notation will be used: SUCC(\( S_i \)) is the successor of the server \( S_i \) in the ring of servers and succ(\( S_i \)) is its successor node in the ring of users.
The control packet embedding the token, contains several fields which are now discussed.

The token type. We recognize two types of tokens, one which visits only server nodes and another one which visits server and user nodes. Following the terminology described in [7] and [8], we call the first type CAP and the second SAP. Most of the fields described now are relevant only for the SAP.

The operation mode. There are two operation modes. In a scheduled mode, a station receiving the SAP token can transmit only the service requests selected by a service request masks (services which are currently offered by the server which has created the SAP). In the non-scheduled mode, a station receiving the SAP is allowed to send any service request.

The residual data count. This field of a SAP is set by the server which has originated the SAP. This field is continuously decremented by a transmitting station with an amount equal to the data being transmitted by the station. A station cannot transmit if the logical entity it intends to transmit is larger than the residual data count found in an arriving SAP.

The function selection mask. It is relevant only in the scheduled access mode. The field is set by the server originating the SAP and contains the set of services the server is willing to provide at that time.

The basic format of a data packet is only slightly different from a traditional one. The difference is that the destination address contains now a flag indicating whether the packet contains a request for service, and in this case, the address field should be interpreted as a service id, or, a selective service request was sent, and the field contains both the service id and the server identification, or the packet contains the result of a service request and then the field contains both the service id and the client's address, or the field contains just an address.

The protocol is subject to conditions C1-C3 presented in section 3 and the CAP cycle time has the properties P1 and P2. The CAP token visits only server nodes. All servers connected to the network have to agree upon:
- A target CAP cycle time, called TCAP in the following and denoted by $T$.

- A way of allocating the available bandwidth among themselves, so that the time server $S_i$ operates in a scheduled transmission mode is bounded:

$$ s_i \leq F_i = f_i \cdot \sigma_i. $$

(4.1)

Let us denote by $\text{CAP}(i)$ the event that server $S_i$ is in control of the channel. This is the time elapsed since $S_i$ has received CAP and the moment it has passed the CAP to its successor in the ring of servers. The $\text{CAP}(i)$ is logically divided into two intervals, one corresponding to the scheduled access and one corresponding to the non-scheduled access. The server $S_i$ performs the following action, subject to condition (4.1) which limits the duration of the scheduled access period:

- It sends out the results of processed service requests, for a total time equal to $r_i$.

- Then it converts the CAP into a SAP and passes it to the successor node, succ($S_i$)

When sending out the SAP packet the server adjusts its fields accordingly. The type field is set to SAP, the operation mode to scheduled access, and the residual data count is set to satisfy condition (4.1). More precisely, the residual data count is set to $(s_i - r_i)$.

Finally, the function selection mask determines which services the station is willing to perform. Rather than providing a full list of all services the server is willing to accept, an alternate scheme is to provide just a modification list describing only the services added or deleted from the previous cycle. An even simpler scheme is to have each server send only its address or id and have the clients keep track what services are offered by every server. Clearly the last scheme is less dynamic than the others.

Each of the $N-1$ stations visited by the SAP is allowed to send data subject to two conditions:

- the service requests buffered at the station match the ones advertised by the server,
the amount of data to be transmitted is smaller than the residual data count in SAP.

To send data, a station removes the SAP, inserts its data packets and finally reinserts an updated SAP with the residual data count decreased by the amount of data sent. Any other station which needs service proceeds in a similar manner. On the other hand, any other server (there are \( M-1 \) of them) belonging to the same service group(s) may remove the corresponding data packets on their way to the server \( S_i \). If no such server exists, all packets are removed by server \( S_i \).

When the SAP reaches \( S_i \), the scheduled period terminates and the server decides whether it will enter an unscheduled mode. The decision is based upon the condition that the CAP has arrived early to server \( S_i \). The measure of this earliness:

\[
a_{e,j} \leq \max(0, T - C_{e,i-1})
\]

is translated into a residual data count. The new SAP with the mode bit set to unscheduled mode, circulates again and this time the content of the selection field is disregarded; any station may pass any request for service, subject only to the residual data count condition.

If no server is willing to provide the service, the original sender will have to remove the offending data. When the server \( S_i \) receives the SAP token, it converts it into a CAP packet and passes it to the next server.

5. PERFORMANCE ANALYSIS


There are fewer results for the performance analysis of distributed systems built around a broadcast system. The analysis of such systems leads to multi-server multi-queue systems as the
ones analyzed by Kohen [5].

The following analysis is an unsophisticated attempt motivated by the desire to gain insight into the system rather than by hope of obtaining exact performance indexes for the system.

An exact analysis of the system described is not possible at this stage and even an approximate analysis is challenging. The first objective of our analysis is the study of the scheduled mode, in order to determine the average CAP and SAP cycle times, and then the average delay experienced by a server and by a client. Then the analysis of the unscheduled mode is carried out.

Simplifications and approximations are introduced in the following at two different levels, the definition of the model and the analysis of the simplified model. Let us consider a network consisting of \( N \) nodes, \( M \) of them being server nodes. For the analysis of the scheduled access, we consider a fully symmetric system. The relevant aspects of this symmetry are:

A1 - When a server is in control it "sees" a set of \( N - 1 \) user nodes (since each server may request services from the others). For all user nodes, the arrival processes are statistically independent Poisson processes with equal average arrival rates of \( \lambda \) requests/sec.

A2 - The services provided by the system are evenly spread among the servers so that each client maps its services to a server following a uniform distribution. \( \lambda M \) requests/sec will be directed to any server by a user node. All services provided by the system have identically distributed communication and computational characteristics. More precisely, we assume that each service request fits into one data packet. The size of all such packets are random variables \( X \), identically distributed for all services offered system-wide, with first and second moments denoted by \( \bar{X} \) and \( \bar{X}^2 \) respectively. Each result fits into a data packet of size \( kX \) where \( k \) is a constant.
A3 - The walk time between two neighbor stations in the ring of users, $w_u$, is constant and the same for each pair of stations. The walk time between two neighbor servers, $w_s$, is also constant. In both cases, the walk time includes the channel propagation time.

5.1 Average CAP cycle time

Let us denote by $T_S$ the average cycle time in the ring of servers, called in the following the CAP cycle time. Let $R$ be the channel capacity and $N_s$ the number of service requests a server in control of the channel has to process.

When one server, say server $S_i$, receives the CAP, it carries out the following actions:

- it uses a time equal to $N_s \cdot k \cdot \frac{X}{R}$ to send out the results for service requests received during the previous CAP cycle,
- it sends out the SAP which visits all other $N-1$ nodes. Each node needs $\frac{X}{R}$ to send its service requests to $S_i$. Then it passes the SAP token to the next node,
- after receiving back the SAP, $S_i$ passes the CAP along to the next server, $SUCC(S_i)$.

Hence,

$$T_S = M \left\{ N_s \cdot \frac{kX}{R} + N_s \cdot \frac{X}{R} + N \cdot w_u + w_s \right\}$$

with

$$N_s = \lambda \cdot \frac{N-1}{M} \cdot T_S$$

Expression (5.2) states that at each other node (there are $N-1$ of them), the service requests addressed to server $S_j$ are generated at a rate $\frac{\lambda}{M}$ for the duration of a CAP cycle (server's vacation). It follows that the CAP cycle time can be expressed as:

$$T_S = \frac{M(N \cdot w_u + w_s)}{1 - \lambda(N-1)(1+k) \frac{X}{R}}$$
We now define the throughput of the network, $S$, as the ratio of the total data arrival rate to the network, to the channel capacity:

$$S = \frac{N \lambda X}{R} (1+k)$$  \hspace{1cm} (5.4)

If we denote

$$S' = (N-1) \frac{\lambda X}{R} (1+k)$$  \hspace{1cm} (5.5)

We see that

$$S' = S \left[ 1 - \frac{1}{N} \right]$$  \hspace{1cm} (5.6)

It follows that (5.3) can be rewritten as:

$$T_s = \frac{M(N \cdot w_u + w_s)}{1 - S \left[ 1 - \frac{1}{N} \right]}$$  \hspace{1cm} (5.7)

Let us now define "the scheduled service time" for each server to be:

$$F_s = \frac{1}{M} T_s$$  \hspace{1cm} (5.8)

or:

$$F_s = \frac{N \cdot w_u + w_s}{1 - S \left[ 1 - \frac{1}{N} \right]}$$  \hspace{1cm} (5.9)

5.2 Delay analysis for the result packet

We approximate now the delay experienced by a result packet generated by server $S_i$, in case of a service request received during an earlier CAP cycle. This delay denoted by $W_s$ has two components:

$$W_s = W_{s,1} + W_{s,2}$$  \hspace{1cm} (5.10)

with:
The average time elapsed from the generation of the result packet to the time the server receives the CAP, $W_{S,1}$, is:

$$W_{S,1} = T_{out}$$

The average time spent by the result packet in the output queue of the server, waiting to reach the head of the queue, $W_{S,2}$, is:

$$W_{S,2} = \frac{T_s}{P_S}$$

To estimate $W_{S,1}$, we define $P_S$ as:

$$P_S = \frac{\lambda}{N} \frac{k}{M} \frac{X}{R}$$

with this definition,

$$T_{out} = P_S T_s$$

is the interval of time, during a CAP cycle, when a server is allowed to send out result-packets.

As shown in Figure 1, the results-packets are produced at random during the time $(1 - P_S)T_s$.

![Figure 1.](image)

Now we have to remember that all service requests have arrived at server $S_i$ in a relatively short period of time, as compared with $T_s$, during a previous SAP cycle initiated by $S_j$. These requests are processed by server $S_i$ according to its scheduling policy. The exact distribution of the time when results-packets were generated, depends upon the arrival process to server $S_i$, and the service process at server $S_i$. We consider a simplification of the analysis and assume that the results were uniformly generated during the time $(1 - P_S)T_s$. Then $W_{1,1}$ becomes:

$$W_{1,1} = \frac{(1 - P_S)T_s}{2}$$

From (5.4) and (5.10) we obtain
Hence

\[
T_S = \frac{M(Nw_u + w_s)}{1 - \rho_S M \left[ 1 + \frac{1}{k} \right]} \tag{5.15}
\]

and

\[
W_{S,1} = \frac{(1 - \rho_S)}{2} \times \frac{M(Nw_u + w_s)}{1 - \rho_S M \left[ 1 + \frac{1}{k} \right]} \tag{5.16}
\]

To obtain \( W_{S,2} \) we must estimate the average delay experienced by the result packets to reach the head of the server's output queue. We follow now a heuristic argument similar to the one in reference [4]. We consider an equivalent network in which the \( M \) server queues are looked upon as a single lumped queue with an aggregated arrival rate and that this system is an \( M | G | 1 \) system. The average delay in an \( M | G | 1 \) system is obtained from Pollaczeck-Khintchine formula as:

\[
W^* = \frac{\gamma'(\mu')^2 \left[ 1 + (C'_S)^2 \right]}{2(1 - \rho')^2} \tag{5.17}
\]

with:

\[
\gamma' = \text{the average arrival rate}
\]

\[
\mu' = \text{the average service time}
\]

\[
\rho' = \gamma' \times \mu'
\]

\[(C'_S)^2 = \text{the coefficient of variation of the service time.}\]

In our case we have:

\[
\gamma = M \left[ \lambda \frac{N-1}{M} \right] \tag{5.18}
\]

\[
\mu' = k \frac{\bar{x}}{R} \tag{5.19}
\]
\[ C_s^2 = \frac{\frac{\langle kX \rangle^2}{R^2} - \left( \frac{k \overline{X}}{R} \right)^2}{\left( \frac{k \overline{X}}{R} \right)^2} \] 

(5.20)

Then:

\[ W' = \frac{\lambda}{2(1-p')} \left[ \frac{\langle kX \rangle}{R} \right]^2 \left[ 1 + \left( \frac{\overline{X}^2}{\langle X \rangle^2} - 1 \right) \right] \] 

(5.21)

Hence:

\[ W_{S,2} = \frac{\lambda(N-1)}{2 \left[ 1 - \lambda(N-1)k \frac{\overline{X}}{R} \right]} \times \frac{k^2 \overline{X}^2}{R^2} \] 

(5.22)

5.3 SAP cycle time

The average delay experienced by a server say \( S_i \) from the moment when it has received the CAP until it generates the SAP, is equal to the time it takes to send out results-packets. This time, \( T_{out} \) is given by expression (5.11).

It follows that the average SAP cycle time, \( T_U \), is equal to

\[ T_U = \left( \frac{T_S}{M} - w_s \right) - T_{out} \] 

(5.23)

or

\[ T_U = \left[ \lambda \times \frac{N-1}{M} (k+1) \frac{\overline{X}}{R} - \rho_s \right] T_S + Nw_u \] 

(5.24)

Then:

\[ T_U = \left[ \lambda \frac{N-1}{M} \frac{\overline{X}}{R} \right] \times \frac{M(Nw_u + w_s)}{1 - S \left( 1 - \frac{1}{N} \right)} + Nw_u \] 

(5.25)

and finally:

\[ T_U = \frac{S \left( 1 - \frac{1}{N} \right) (Nw_u + w_s)}{(1+k) \left[ 1 - S \left( 1 - \frac{1}{N} \right) \right]} + Nw_u \] 

(5.26)
5.4 The delay analysis for a service request packet

We assume that each server has the same behavior from one CAP cycle to another. This analysis will be carried out in two different cases. Case one corresponds to the optimal case as far as this delay is concerned, namely, every server provides all services available. Case two corresponds to the opposite, namely each service is available on only one server.

In both cases, the delay $W_U$ has two components namely:

$$W_U = W_{U,1} + W_{U,2} \quad (5.27)$$

with

$W_{U,1}$ = the average time elapsed from the generation of the service-request packet until the station receives a SAP, which allows the user station to transmit the packet,

$W_{U,2}$ = the average time spent by the request packet waiting to reach the head of the queue, measured since the SAP authorizing the transmission has arrived.

$W_U^{(1)}$ will be the total delay corresponding to the first case and $W_U^{(2)}$ the one for the second case.

Case 1.

Let us define:

$$\rho_U = \lambda \frac{1}{M \bar{X}} \quad (5.28)$$

Then $\rho_U \cdot T_U$ is the interval of time during a SAP cycle, when the user is allowed to send its request packets. Such packets arrive at the user node at random points in time during a period of length

$$(1 - \rho_U)T_U + T_{ow}$$

Following the same arguments presented in reference [4, page 205] we consider that the arrivals are distributed uniformly throughout this period. Hence

$$W_U^{(1)} = \frac{(1 - \rho_U)T_U + T_{ow}}{2} \quad (5.29)$$

with $T_U$ and $T_{ow}$ expressed in terms of $\rho_U$ as:
To determine $W_{U_1}$, we use again the heuristic argument from sections (5.2) and the expression (5.17). Now we use the notations:

$$\lambda' = (N-1) \frac{\lambda}{M}$$

$$\mu' = \frac{X}{R}$$

$$\rho' = (N-1) \rho_U$$

(5.32)

and

$$(C'U)^2 = \frac{X^2}{(X)^2}$$

It follows that

$$W_{U_2} = (N-1) \frac{\lambda}{M} \frac{R^2}{2(1-(N-1)\rho_U)}$$

(5.33)

Finally, the total delay is:

$$W_{U_1} = W_{U_1} + W_{U_2}$$

(5.34)

Case 2

In this case, a client has a window of size equal to $\rho_U \cdot T_S$ during an entire CAP cycle. During this period it is allowed to send out the service requests buffered at the station. Using the same arguments as before, we can express $W_{U_2}$ as:

$$W_{U_2} = \frac{1 - \rho_U \times T_S}{2}$$

(5.35)

with $T_S$ given by (5.7), $T_U$ by (5.26) and $\rho_U$ by (5.28). It can be easily seen that $W_{U_2} = W_{U_2}$. 

\[ T_U = (N-1)\rho_U \times \frac{M(Nw_u + ws)}{1 - S \left[ 1 - \frac{1}{N} \right]} + Nw_u \] 

(5.30)

\[ T_{out} = \rho_U k(N-1) \frac{M(Nw_u + ws)}{1 - S \left[ 1 - \frac{1}{N} \right]} \] 

(5.31)
5.5 On the analysis of the unscheduled access

When the system described above is fully symmetric and subject to medium communication and computation load, the unscheduled access is insignificant. But it becomes increasingly important as the asymmetry of the system increases. Also, when the system is computation bound (all servers are above the low water mark defined in Section 2), the dominant mode is the unscheduled mode.

Let consider only the case when the system has such a load that \( S < 1 \) and all servers are below the high water mark. In this case we are able to guarantee that any service request will be delivered to a server in the service group within a CAP cycle time, for any level of system asymmetry. This means that in the worst case the waiting time of a service request has \( 2T \) as an upper bound.

6. CONCLUSIONS

The paper describes the architecture of a distributed system based upon a functional communication model with communication initiated by the servers and by the users. An approximate performance analysis of the system is carried out.

LITERATURE


