ON THE CALCULATION OF ACCESS CONTENTION FOR SHARED RESOURCES IN MULTIPROCESSOR CONFIGURATIONS, A GENERAL STRAIGHTFORWARD APPROACH

Georg Daisenberger
Planning Department
Siemens A.G.
Munich, West Germany

A procedure is developed, which approximately calculates the access contention for complex resource configurations. Within the range of practical interest the procedure itself can be approximated by a relatively simple formula. The formula can also be adapted to form a reasonable approximation for closed network solutions.

1. PROBLEM AND MOTIVATION

Load sharing is an effective method of increasing the capacity of an SPC system many times over. This operating mode will, however, lead to collisions when a processor wants to access the state data of the system, which necessarily exist only once. As a consequence, the accesses may be delayed by a certain "waste" runtime which results in the gain of capacity being reduced by a certain amount through contention.

The question is: How to determine the loss in capability? It posed itself in connection with the Coordination Processor of the EWSD system.

The problem, of course, is not new. It makes itself felt especially in connection with analogous conflict phenomena occurring in commercial data processing systems, and the relevant literature offers various solutions based on closed queuing network models. However, there is a basic difference: The processor in a switching system must not be loaded with tasks to the point of full saturation, and therefore the application of closed network models is not a legitimate approach.

In the following a calculation procedure (called "calculus") is developed on a pragmatic-heuristic basis, which takes into account all relevant parameters and arbitrary conflict configurations. As a result the ratio of the additional ineffective load to the utilized load (the "relative surplus" for short) is obtained. We may interpret it as the additional capacity which would be needed to cover the waste runtimes. The relative surplus is thus an indirect measure of the loss of capacity.

The calculus yields an approximation for the case of negative exponentially distributed (for short: exp) accessing times, and its application shall be restricted to a range of practical interest (i.e. for a surplus not exceeding approximately 20%-25% overall). Comparison with simulation results has, however, revealed that a high accuracy is attained even beyond this range. Theoretical arguments explaining this behavior will be discussed in detail.

An empirical analysis of the relative surplus as a function of its parameters reveals that it can be approximated with surprisingly high precision by a very simple formula.
2. RATIONALE OF THE CALCULUS

2.1 Terminology

Speaking in generalizing terms, the particular switching task of the processor is considered as a job. The given section of common memory which is to be accessed is generalized as a resource (RSU for short), which the job requests and, after a certain delay, holds. The duration of an access consists of the holding time including the preceding waiting time.

The RSUs representing the access targets may be "configured arbitrarily" with respect to the sequence of accesses. Some subsets may constitute "serial configurations", which means that they are accessed in a sequential order. Other subsets may constitute "parallel configurations", which means that they are accessed at the same point in time, but alternately with a predetermined probability. For the general case, we refer to each RSU simply by a number \( r = 1, 2, \ldots \) (Fig. 2.1). The following quantities, thought of as representing statistical averages, are taken into consideration:

- \( \tau \): Runtime of a job (without waiting time)
- \( h_r \): Holding time of RSU \( r \) per job
- \( w_r \): Waiting time for RSU \( r \) per job

Note that \( h_r \) and \( w_r \) are not necessarily the holding time and waiting time, respectively, of just one access. These quantities may, for instance, result from 2 or 0.5 accesses per job. In our simplified model, however, only \( h_r \) will be relevant.

The capitals \( T \), \( H \), and \( W \) will represent random variables corresponding to \( \tau \), \( h \), and \( w \) in the sequel. \( N \) represents the number of processors, and \( \lambda \) the arrival rate of jobs. The utilization of the processors thus is \( \phi = \lambda \tau / N \).

Using

- \( p_r := h_r / \tau \), "relative load of RSU \( r \)"
- \( u_r := w_r / h_r \), "relative waiting time for RSU \( r \)"

the relative surplus can be represented as

\[ b_r = u_r / \tau = (w_r / h_r) \cdot (h_r / \tau) = u_r \cdot p_r \quad (2.1/1) \]

The factor \( p_r \) in this formula follows immediately as the load on RSU \( r \), related to the total load on the processors. How the "relative waiting time" \( u_r \) can be determined (the "heart of the calculus") will be explained in the next section.

Apart from a negligibly small mutual interference, all the \( b_r \) are considered to be additive. This means that the total relative surplus \( b \) of the entire system is obtained simply by
adding all the $b_r$. The (relative) occupancy of the processors follows from this as $\eta = \varphi \cdot (1+b)$.

2.2 Calculating the Waiting Time for an RSU

Since only one particular RSU is under consideration, the indicating subscript $(r)$ is omitted. If not stated explicitly, all sums apply for an index running over $0, ..., N$. The calculation procedure comprises four steps:

Step 1: Initially, we assume the RSU to be modified in such a way that it works as a multi-server system, which means that collisions will lead to multiple holds instead of to delays. We take a "snapshot" at an arbitrary point in time, supposing a stationary equilibrium, and now ask for the probability distribution of the number ($Y$) of RSUs held simultaneously. The following hypotheses are put forward for this purpose:

(H1) The distribution of the number of jobs in progress simultaneously ($X$) is known (we assume an $M/M/N$ for simplicity)

(H2) The probability of encountering a job holding the RSU at snapshot time ("holding probability") is $h/\tau = p$.

(H3) The holding of the RSU in question takes place stochastically independent of the state of all the other jobs.

From this, the distribution of $Y$ results as a superposition of binomial distributions:

$$U_k = \sum_{n=k}^{N} P_n \binom{n}{k} p^k (1-p)^{n-k} \quad (k=0, ..., N)$$

where $U_k=P(Y=k)$, and $P_n=P(X=n)$. Assuming an $M/M/N$, this superposition can be calculated immediately by means of a straightforward algorithm (see the appendix). Note that (H2) is confirmed by fundamental theorems.

**FIGURE 2.2**

Characteristic behavior of the arrival rate of requests ($\alpha_k$) in relation to the occupancy of the processors ($\eta$).

Step 2: We ask for the properties of an arrival process which may generate a distribution as calculated in step 1. The following simplifying assumptions are made for this purpose:

(H4) The arrival process is poissonian, but with a state-dependent arrival rate. The "state" is represented by the number of simultaneously existing accesses (which are represented only by holding times in this case).

(H5) The holding time of the RSU is negative-exponentially distributed (exp for short).

The conditional arrival rates ($\alpha_k$) can now be calculated by simply using the equilibrium equations.
\[ U_k = \frac{k}{h} U_k \quad (k=1, \ldots, N) \] (2.2/2)

with known state probabilities \( U_k \) in this case.

Fig. 2.2 demonstrates the characteristic behavior of these \( \alpha_k \) in relation to the occupancy \( n \) (which is equal to \( q \) at this point of calculation). For reasons of graphic representation, the load of the RSU (\( N \_ p \)) is held constant throughout the diagram. As to be seen from the diagram, the arrival stream for \( q-1 \) turns over into an "Engset stream" with \( \alpha_k \) being proportional to \( N-k \), and for \( q-0 \) tends to a "truncated Poisson stream" with \( \alpha_k \) being constant for \( k=1, \ldots, N-1 \), and zero for \( k=N \).

Step 3: We return to reality: We now treat the RSU as a single server, with a waiting field organized in a FIFO discipline, and apply the arrival stream calculated in step 2. From this, a distribution \( V_k=P(Y^+ = k) \) is obtained, where \( Y^+ \) represents the number of accesses in progress. The \( V_k \) are defined by

\[ V_{k-1} \alpha_{k-1} = \frac{1}{h} V_k \quad (k=1, \ldots, N) \; , \; \Sigma V_k = 1 \] (2.2/3)

The relative waste load \( u=w/h \) now can be determined by solving

\[ (1+u) P(Y^+ > 0) = E[Y^+] \] (2.2/4)

for \( u \). The equation is obtained from the fundamental relations \( (w+h) \alpha = E[Y^+] \) and \( \alpha = P(Y^+ > 0) \) (\( \alpha \) representing the average arrival rate), where the second relation allows \( \alpha \) to be replaced by \( P(Y^+ > 0)/h \).

Step 4: Changing from a multi-server to a single-server RSU, however, we shall inflict some distortion upon the load balance, which initially was fitted to the situation in a multi-server RSU. We compensate for this effect by raising the arrival rates. We assume for this purpose:

(H6) The mutual relations of the state-dependent arrival rates are invariant, i.e. all arrival rates are changed by the same factor.

The factor is defined by the condition that, in an M/M/1, the load is equal to the probability of the server being busy, which means that

\[ P(Y^+ > 0) = N q p \] (2.2/5)

It can be determined by iteration (repetition from step 3).

3. THEORETICAL ARGUMENTS

3.1 Introductory Remarks

The calculation procedure developed above obviously implies several inconsistencies. Carefully considered, it can be said to be an approximation, which applies for an approximating "substitute model". This model differs from the reality in that the waiting time is assumed to take place outside of the processors, and thus does not affect the runtime of a job. Fig. 3.1 may elucidate this fundamental difference.

For this reason, the calculus produces results also for load conditions beyond the real system's point of saturation (which obviously indicates a logical conflict). Nevertheless, results obtained beyond saturation can be considered as hypothetical, but valuable information.

In order to establish a correspondence to the reality, we go on to investigate the actions of the calculus on the basis of a concrete job schedule. For reasons of transparency, it is assumed that the job needs exactly one access which is effected in coincidence with its start. The job thus passes through a holding time (\( H \)), and a residual time (\( T_0 \)). In reality, \( H \) is
preceded by a waiting time \((W)\). Note that all these capital symbols are assumed to be random quantities, and the subscripts are omitted.

\[
\alpha_k = \frac{(N-k)}{\tau_0} \quad (k=0, \ldots, N, \text{for } \eta \rightarrow 1).
\]  

For the complementary case of \(\eta \rightarrow 0\), it is true, an analysis based on formal rules is not so easy. But (for space saving reasons) we can conclude by means of plausibility that, except in the case where \(N\) accesses are in progress, with a probability close to 1, an idle processor is always available for arriving jobs, which means that the arrival rate of requests tends to \(\lambda\). For \(N\) accesses in progress, however, a job start obviously is impossible because an arriving job is queued, and a terminating job has no queued successor. From this results

\[
\alpha_k = \lambda \quad (k=0, \ldots, N-1), \quad \alpha_N = 0 \quad (\text{for } \eta \rightarrow 0). 
\]  

3.2 Discussion of Calculation Errors

A first source of calculation errors is to be found comparing the prerequisites for step 1 and 2: Assuming an \(M/M/N\), and an \(N\)-fold RSU, the result of step 1 corresponds to an exp \(T_0\) and \(H\) being arbitrarily embedded within \(T\) (e.g. on top). But \(H\) and \(T_0\) are mutually dependent, and at least one of them is not exp. For step 2, on the other hand, it is necessary to postulate \(H\) and \(T_0\) as exp and mutually independent. As a consequence, the processors no longer constitute an \(M/M/N\). The calculation error caused by this discrepancy, however, has proved to be of relatively little significance.

The main source for calculation errors rather arises from step 2 by ignoring that the occupancy is increased by the surplus \(b\). Compared with the real arrival stream, the calculated arrival stream thus tends to a Poisson stream, which effects a pessimistic tendency. If we wish to take steps to correct this aberration by basing the calculation on the real \(\eta\), however, we have to take into account that the job's runtime is now \(T + W\), instead of \(T\), and the relative load of the RSU is reduced to \(p^+ := p/(1+b)\) for that reason.

Let \(\varphi(\varphi, p)\) be a function representing the result of the calculus for a certain RSU in consideration. Then the correction is equivalent to finding \(b\) as a solution of

\[
b = \varphi(\varphi \cdot (1+b), p/(1+b)) \quad ,
\]  

i.e. \(b\) can be determined by iteration.
It is not too difficult to expand this approach to be suitable for a complex configuration of RSU\(_r\) \((r=1,2,\ldots)\). For this application, \(b\) is to be taken as the sum of all \(b_r\), where \(b_r\) results as \(\gamma_r(\nu(1+b),\frac{P_r}{(1+b)})\). and \(\gamma_r\) applies for RSU\(_r\).

The gained improvement, however, seems not to be worth the additional efforts, since \(b\) is reduced only by about 5\% to 15\%, depending on the actual case. Nevertheless, the formula (3.2/1) serves as an important tool for the verification of the calculation results.

Moreover, in the light of step 2 being based on the real \(n\), it is now possible to interpret the raising of the \(\alpha_k\) - initially introduced simply for the pragmatic purpose of re-establishing a correct load balance - in correspondence with a concrete time schedule. The consideration, however, is limited to the marginal cases of \(\nu\) tending to 0 or 1. Doing this, we have to keep in mind that step 2 now calculates an arrival stream based on a residual time \(\tau_o+:=\tau_o+w\) instead of \(\tau_o\).

Since for \(\nu\rightarrow 0\) the arrival stream of requests turns into a Poisson stream, step 4 will leave the \(\alpha_k\) unchanged. For \(n\rightarrow 1\), however, the \(\alpha_k\) follow equ. (3.1/1) with \(\tau_o\) being replaced by \(\tau_o+:=\tau_o+w\). An increment by a common factor, say \(f\), thus is equivalent with \(\tau_o\) being divided by \(f\). Since (as can be shown) \(f\) is equal to \(\tau_o+:/(\tau_o+\nu-w)\) in this case, the rise corresponds simply to a shortage of \(\tau_o+\nu\) by the mean waiting time \(w\), i.e. step 4 re-establishes the \(\alpha_k\) in correspondence with the correct residual time \(\tau_0=\tau_0+:w\).

The calculus thus turns out to be an "iteration procedure which is aborted just after the opening step".

4. VERIFICATION

Checking the results of the calculus for validity can be accomplished by a comparison with simulation results. As an example Fig. 4. shows the total relative surplus versus the utilization \(\nu\) for \(N=8\) processors and \(M=5\) parallel-configured RSUs, which are accessed alternately with the same probability. In the literature, such a configuration is said to be an "RSU with granularity \(M\)". As the relative load of each RSU, \(p_r=0.1\) \((r=1,\ldots,5)\) is assumed, which means that the mean holding time \((h_r)\) amounts to 50\% \((1)\) of the mean runtime.

![Figure 4](image_url)

**FIGURE 4.**
Calculated (---) and simulated (\(\square\)) relative surplus \((\nu)\) versus the utilization \((\nu)\) of the processors.

The simulation is based on a Monte Carlo method assuming a Poisson stream of jobs, and a job schedule with \(\exp H_r\) and \(T_0\) (arranged as in section 3.2). 100000 jobs are involved. The deviation of the point \(\nu=0.69\) is caused by saturation.

But in the present case, a second approach is also possible:
Keeping \( \phi \cdot (1+b) \) constantly equal to 1, equ. (3.2/1) applies for the case of the processors being saturated (by utilized plus ineffective load), and thus must correspond to a closed network model. Considering an RSU with granularity \( M \), this is, in fact, verified by inspection (which suggests that the corrected calculus is an exact solution in this case).

5. APPROXIMATION

The \( u_r \) resulting from the calculus (section 2.2), within the range of practical interest (say \( N \leq 10, \ p_r \leq 0.1 \)) can be approximated by

\[
\begin{align*}
    u_r &= \psi_r(\phi, p_r) := \frac{G \cdot p_r \cdot (N-1)}{1 - F \cdot p_r \cdot (N-2)} \quad (N \geq 2) \\
    G &= (1.5 - 0.5 \phi) \phi, \quad F &= (1.05 - 0.1 \phi) \phi .
\end{align*}
\]

For the hypothetical "worst case" \( \phi = 1 \) (where \( G = 1, \ F = 0.95 \)), the approximation by \( \psi \) is practically perfectly conform with the calculus (i.e. graphs optically appear to be congruent). For \( \phi \) decreasing to values <1, an increasing loss of accuracy is to be observed, reaching a "worst" (but, nevertheless, reasonable) quality at about \( \phi = 0.3 \). Using equ. (2.1/1), \( b_r \) results simply by multiplying \( u_r \) with \( p_r \).

For the special case of an RSU with granularity \( M \), the total relative surplus thus simply adds up to \( M \cdot p_r \cdot u_r \). For the situation of the processors being saturated (i.e. \( \phi \cdot (1+b) = 1 \), as postulated earlier in section 4.), a reasonable approximation corresponding to the closed network solution is attained by using \( G = 1 \), and \( F = 0.86 \).

APPENDIX

In order to develop a straightforward calculation of the distribution \( V_k \) assume

\[
\begin{align*}
    J_n &= a^n / n! , \\
    L_k &= \sum_{n=0}^{N} J_n \left( \begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k} \quad (k=0, \ldots , N) .
\end{align*}
\]

Substituting \( J_n \) in (A/2) by (A/1) then yields

\[
\begin{align*}
    L_k &= \frac{(ap)^k}{k!} \sum_{n=k}^{N} \left( \frac{a(1-p)n-k}{n-k}! \right) \quad (k=0, \ldots , N) ,
\end{align*}
\]

from which after substituting \( n-k \) by \( i \) is found that

\[
\begin{align*}
    L_{N-k} &= \frac{(ap)^N}{N!} \left( \frac{N!}{(N-k)!} \right) \sum_{i=0}^{k} \frac{(a(1-p)i)!}{i!}
\end{align*}
\]

This formula enables \( L_{N-k} \) to be calculated recursively for \( k = 0, \ldots , N \). In the case that the processors constitute an \( M/M/N \), the \( P_n := P(X=n) \) can be represented by \( P_n = C \cdot J_n \) (\( n = 0, \ldots , N-1 \)) and \( P_N = C \cdot J_N / (1-p) \), with \( a = N \phi \), and \( C \) being a normalizing constant. The distribution \( V_k \) follows from this as

\[
\begin{align*}
    V_k &= C \cdot (L_k + J_n \frac{\phi^N}{1-\phi} \left( \begin{array}{c} N \\ k \end{array} \right) p^k (1-p)^{N-k})
\end{align*}
\]