Under and over sharing equations relating throughput capacity, resource-sharing level and the source of thrashing in multi-thread systems

Research Report No: 2001-694-17

James Bradley
Department of Computer Science
University of Calgary,
Calgary, Alberta, Canada

Abstract We propose a new, non-empirical, sharing-level version of the sharing equation, which relates throughput capacity to the level of resource sharing in a system. This sharing-level version uses a sharing-level independent variable, as opposed to a sharing-algorithm time variable in an earlier version of the equation.

This sharing-level version is more fundamental, given that the resource-sharing level in a system is a more fundamental performance parameter that is more easily measured.

There has to be two distinct equations, however. One is for the case of *under sharing*, the under-sharing equation, where the sharing level is such that the throughput is less than the peak possible with an optimum sharing level. With the under-sharing equation, throughput capacity climbs with increasing sharing level. The other equation is for the case of *over sharing*, the over-sharing equation, where the sharing level exceeds the optimum sharing level. With the over-sharing equation, throughput capacity decreases with increasing sharing level.

The over-sharing equation also reveals the true source of thrashing, when the system is operating under extreme over sharing conditions

A single empirical sharing equation is also presented, which works reasonably well for both over and under sharing, at the cost of a somewhat less than true peak throughput capacity.

Key words *performance, resources, resource sharing, sharing procedure, system, threads, throughput capacity.*

1.0 Introduction

In this paper we present a new version of the fundamental resource-sharing equation. This version uses the resource-sharing level present in the system as the dependent variable, and gives throughput capacity as a function of sharing level. We define what we mean by sharing level later in the paper, but for now it can be taken as a measure of the amount of resource sharing present in the system.

We consider this equation to be an improvement over a resource-sharing equation presented in a previous paper [3]. That version related throughput capacity to the execution time of the resource-sharing procedures needed to share the resources among threads of execution.

The new version of the equation actually splits into two equations. One is for the case of under sharing, *the under sharing equation*, where the sharing level is such that the throughput is less than the peak possible with optimum sharing level. With the under-sharing equation, throughput capacity climbs with increasing sharing level. The other equation is for the case of over sharing, *the over sharing equation*, where the sharing level exceeds the optimum sharing level. With the over-sharing equation, throughput capacity decreases with increasing sharing level. In this paper we shall derive both equations. We also present a single empirical equation, which works reasonably well for both over and under sharing.

The research leading to these resource-sharing equations was inspired by phenomena in complex computer systems [4, 10], and it is expected that it is in the computer systems arena that the equations will throw the most light. Nevertheless, the equations do cover systems phenomena outside of the computer arena as well, particularly in industrial production systems [7], as will be exemplified in the paper. Thus, the equations express a unity of principle in a great diversity of system phenomena.

Mindful of the fact that most discussions of systems are weakened by lack of precise definitions and units of measurement, in this paper every attempt has been made to ensure that concepts are clearly defined, and that symbols appearing in the equations denote quantities that are measurable and expressible in clearly defined units.

Definition of a system

A system is considered to be any entity that converts inputs to outputs, and which may be composed of subsystems, connected either in series or in parallel, each of which is also a system. We also consider a system to be an entity that functions under the direction of one or more agents, usually human, and which employs essential resources R, often portable, enabling it to convert a set of inputs N, informational or material or both, to a set of outputs U per unit time, so that U is the system throughput. If the maximum value for U is I, then I is said to be the throughput capacity of the system, measured as the units of output per unit time when the system is operating at maximum capacity [8, 9]. In general, if the system is operating at a fraction f of throughput capacity I, then the throughput is U = fI.

In some cases there will be more than one type of throughput product. In that case we can declare throughput capacity of one of them as being the main throughput product capacity I, in units of product per time period. It can then be assumed that each other product throughput capacity is a byproduct capacity that is a function of I, in most cases a linear function. Accordingly, in this paper, we deal only with I, considered to be either the only or the main throughput capacity.

Measure of resources

The units of both R and P are units of anything of value to a human agent. Thus R could be measured in dollars, marks, printers or printer equivalents, microprocessors or microprocessor equivalents, or even, where the resources are mostly human, such persons as programmers. Also, since a resource can be valued by the human effort or work, or work equivalent, required to produce it, and since work is a measure of energy, then ultimately resources R can be measured in units of energy [11, 12].

Overview of the resource sharing equations

The following is the under-sharing equation governing the relationship between throughput capacity and sharing level, for the case of under sharing:

$$I = KR[1 + s] \qquad \textit{resource under-sharing equation}$$

Here I simply increases linearly with the level of sharing. This simple resource-sharing equation is the most useful one in practice. In the above resource under-sharing equation, a sharing level of s means that the resources are processing s + 1 threads at a time. Thus when s = 0, it means that is there is no resource sharing, and the equation reduces to I = KR, which we would expect.

The following is the over-sharing equation for over-shared throughput capacity as a function of sharing level s:

$$I = RK(1 + f)[p + a(f)]/(p + a(s))$$

resource over-sharing equation

where R is the system resources, K is a constant, f is the sharing level at which throughput capacity peaks, p is the actual (possibly average) processing time per thread, a(s) is sharing administration time per thread, and s is the actual sharing level—and a sharing level that causes over sharing. When s is f, the level at which throughput capacity peaks, the above equation reduces to:

$$I = RK(1 + f)$$

as we would expect.

2.0 Coordinated, inclusive resource sharing in systems

Before we attempt to derive the above equations, we give a brief explanation, and then two examples, to clarify the nature of the resource sharing we are dealing with in this paper.

Suppose that we have two subsystems sysA and sysB, each with resources A and B, and suppose the two systems are operating in series, with the output of sysA serving as the input of sysB. If sysB is limiting, that is, operating at full capacity, then throughput capacity may still be increased, by increasing both B and A in proportion

Alternatively, we may be able to increase B but keep A constant, and allow A to increase throughput capacity by participating in resource sharing instead. There are two types of resource sharing operation, the simple one and the complex one.

The simple one involves uncoordinated, exclusive-allocation sharing of resource A among complete task threads. The chief characteristics of this simple case are that interleaving of subtasks within task threads is not involved, nor are time-consuming resource-sharing procedures.

In contrast, the complex kind of resource sharing does involve sharing of resource A among subtasks from different task threads, and we call it coordinated, inclusive-allocation resource sharing. Thus interleaving of subtasks within task threads is involved, as are complex resource-sharing procedures [1, 2].

In both cases there is resource sharing, but in each case the nature of the sharing is very different, and the difference is vital for understanding the operation of complex systems, and the risks that complex resource sharing gives rise to, transaction processing risks in particular [5]. We will first briefly examine a typical example of the first, simpler kind of resource sharing, as a lead into an example of the more complex kind of sharing. This should promote the reader's understanding of the differences between the two types of resource sharing in systems.

In the rest of this paper we assume the fundamental systems proposition that when no coordinated resource-sharing procedure is involved, throughput capacity I relates to systems resources R according to I = KR, where K is a constant. This expression states that, if we increase resources R (in a valid manner), then I will increase linearly with R. For example, if we construct a system that is an m-fold replica of the original system, with resources mR, then throughput capacity will be mI. We take I = KR as axiomatic for valid changes to R.

Example of simple and complex sharing of resources

Suppose we have an automobile manufacturing production line. There are two subsystems in series, a paint spray facility (sysA), followed by a polishing facility (sysB). We suppose it takes

30 minutes for a car to go through the paint facility, and two hours for a car to go through the polishing facility. If cars feed through these two units in a steady stream, we will get one car through the combined system every two hours. Note that after the paint facility has been in use for 30 minutes, it sits idle for the next 90 minutes.

The polishing facility (sysB) is the limiting, or full-capacity, resource, and there is a mismatch between the throughput capacity of the paint facility (sysA), which is one car every 30 minutes, and the polishing facility, which is one car every 2 hours.

We can remedy this obvious mismatch by adding three more polishing units, for a total of four, but keeping the single paint facility. Now the output from the paint facility simply feeds into each of the 4 polishing facilities in turn. As each car emerges from the paint shop, after 30 minutes of painting operations, it goes to the polishing shop that has just finished a car.

We thus share the paint facility in a simple way among cars going to the four polishing facilities next on the production line. This is the case of uncoordinated, exclusive-allocation sharing of resource A among complete 30-minute long, painting-task threads. A complete painting-task thread is the sum of all the painting subtasks in sequence needed to paint a single car.

It might be thought that this is the limit to possible sharing. However, this is not the case. A great deal of improved throughput capacity by means of a different kind of resource sharing may be possible. The following case of coordinated, inclusive-allocation resource sharing shows how, and needs to be pondered carefully.

Incidentally, the fact that the example below is very obvious and even mundane, and is about cars and painting shops, is irrelevant. Large numbers of equivalent examples can easily be constructed for very different systems, all of them very similar to the paint shop example below. The example after this one, involving computer operating systems, among the most sophisticated of systems [6,10], is almost exactly equivalent, and involves exactly the same kind of resource sharing.

Suppose now that the 30 minutes a car spends in the paint facility are as follows: The car gets three coats of paint, each coat taking ten minutes. The ten minutes for each coat involves one minute of actual painting, and then nine minutes of drying (assume it is fast drying paint). The sum of these subtasks in sequence over 30 minutes is the painting-task thread of painting subtasks.

Obviously, during the 30 minutes the car spends in the paint facility, it is being painted for only three minutes. We could assume that the actual painting is done by a robotic machine that is quite expensive.

The paint facility allows only two cars through per hour when it is allocated to the entire 30-minute task of painting a car, that is, to the entire thread of subtasks. But the thread of subtasks (particularly the subtasks involving adding a coat of paint), making up the painting task for a given car, could be interleaved with the threads for other cars.

We could thus proceed alternatively as follows: For the first coat of paint, once car-1 has been painted, it could be instantly whisked away from the painting robot to dry, as another car-2 is instantly whisked into position for painting, and is then painted for one minute. Then car-2 is whisked out, and car-3 is painted for one minute, and so on, until car-10 is painted.

At this point the paint for car-1 has dried and car-1 is whisked back into position for its second coat of paint for one minute, followed by car-2, and so on. After car-10 has received its second coat, car-1 is whisked into position for its third coat, and then car-2, and so on. Meanwhile the third coat on car-1 is drying, and after nine minutes, just when car-10 has finished getting its third coat, car-1 now goes to a polishing facility.

We can see that with this way of operating, we can put ten cars through a single paint facility (sysA) every thirty minutes, or 40 cars every two hours, instead of the 4 cars per two hours previously. This is a very large increase in throughput capacity. Of course, we will have to increase the number of polishing facilities (sysB resources B) to match the huge increase in

capacity of sysA. We will now need 40 polishing facilities to match one paint facility, instead of the previous 4, but, if we arrange for that, we will have an impressive ten-fold increase in throughput capacity for the total system.

We also need a complex sharing procedure to make sure that this juggling act does not go awry on us. One small error, and a car may be painted five times instead of three, or only two times, or three times all right—but with the wrong coatings. We shall have more to say about the sharing procedure presently.

Now suppose that originally there had been plenty of polishing facility capacity, perhaps because it is cheap, say with capacity for at least 40 cars per two hours, but only one (very expensive) robot paint facility, outputting a painted car every thirty minutes. Throughput capacity of the total system is therefore only 2 cars per hour. If we now introduce the complex sharing procedure described above for the paint facility, throughput capacity will jump ten-fold to 20 cars per hour—without touching the resources of the system. Such large increases in throughput capacity are the typical result of introducing resource sharing involving interleaving of the subtasks making up task threads.

We saw that we need a sharing procedure, or sharing algorithm, to make all this work. In the example, we assumed that each car would be whisked in and out of the painting area almost instantaneously. In practice, this will not be the case. There will be some time lost by the robot painter while a car is being whisked into position for painting, and also while it is being removed from the painting position afterward. In the example, let us suppose this time takes 15 seconds.

It will still take 1 minute to paint a car and 9 minutes for the coating to dry, so in that ten minutes only 8 cars can now get a coat. For the 8 cars, we need eight minutes for the painting, and 8 x 15 seconds, or 2 minutes of time, for moving the cars in and out, that is, for the sharing-procedure administration time. We thus need 8 + 2 minutes, or 10 minutes for painting and moving 8 cars for one coat of paint.

Our throughput capacity in practice would therefore be increased only to 16 cars per hour instead of the 20 if it took zero time to move the cars in and out, that is, if there was zero time during which the painting resource is allocated to the operation of the sharing procedure. But in practice, a shared resource is never allocated to operation of the sharing procedure for zero time. Some of its time is always consumed in the administration of the sharing operation.

This paint shop example therefore illustrates the basic principle: Operation of a sharing procedure, with interleaving of task threads, will increase the system throughput by a very great deal in theory, but in practice the amount of increase will be reduced by the time consumed, or lost, by the painting resource due to administration of the sharing procedure.

Computer operating systems example of complex sharing of resources.

Now we will look at a second example of sharing the subtasks of a task thread, this one very important, and one that is widely used with computers to improve performance [1, 6,10].

Consider a computer system with a single slow input device (A), a single fast processor/memory unit (B), and a fast, very large capacity data storage disk. The system has a manual subsystem (sysA), in which data is entered by an operator at the slow input device. Another subsystem (sysB) transfers this entered data, by means of the processor/memory unit, to storage on the disk.

The input device is used by only one operator at a time. It takes an operator ten minutes to perform a complete task.

During an operator's use of an input device, input data (i.e. a "job load" of data) is entered and converted via processing to information output on the disk. During a 10-minute session, the computer processor/memory unit is exclusively allocated to the specific input device. With the input device in use all the time, system throughput capacity must be 6 job loads of data per hour, since each session with the input device takes 10 minutes.

At this point, the processor/memory unit B is being shared in an uncoordinated manner among the job loads, with exclusive allocation of B to each job load of data. In other words, a specific job load must finish being processed by the processor/memory unit before the processing of a new one can begin. Furthermore there is no sharing of processor/memory unit B among the subtasks of each thread of subtasks that make up processing of each job load. This is the case of uncoordinated exclusive-allocation sharing of resource B among complete task threads.

We could now double the number of input devices of resource A to two, to enable 12 job loads of data to be entered in an hour. But, if we did, further throughput capacity increase would not occur, because the processor/memory unit must be allocated to entry of a job load at an input device for 10 minutes, and there are only six 10-minute periods in an hour. Either one of the two input devices will never be in use, or each input device will be operating at half capacity, perhaps three times per hour for each input device, assuming there is exclusive allocation of the processor/memory resource B to a job load. System sysB is therefore limiting, at full capacity, and the only way to increase throughput capacity would seem to be to increase both the number of input devices and processor/memory units.

Alternatively, suppose the processing of a job load, as is usual, consists of short processing bursts. Suppose each burst lasts 60 milliseconds on average, and 100 bursts are needed to process a job load. So 6,000 milliseconds, or 6 seconds, of actual processing time, out of 10 minutes (or 600 seconds) of elapsed time, are needed per job load of data. In other words, when the sum of the operations needed to process a job load of data are laid out in time, in sequence, we have a thread of activity, 600 seconds long, consisting of 100 bursts of processing of the job load, each taking 60 milliseconds, each separated in time from the subsequent burst by 5,940 milliseconds, or 5.94 seconds, or just under 6 seconds.

This is similar to the car painting example given earlier, where a painting task was a thread of activity that was 30 minutes long, and consisted of 3 painting bursts, each lasting 1 minute, and each separated in time from the next by 9 minutes.

In the computer case, by interleaving the processor/memory processing of the subtasks for different job loads1, that is, by coordinated sharing of processor/memory resource B inclusively among the subtasks of multiple job loads, we could process up to 594 extra job loads per hour on top of the 6 that renders B limiting, for a total of 600 per hour. Since its takes the processor only 6 seconds of total time to handle a job load, it must have a throughput capacity of 60 x 60/6, or 600 job loads per hour.

This 100-fold increase from 6 per hour is spectacular (because of the enormous computer processor speed), compared to the increase in throughput capacity that can normally be obtained by coordinated sharing with inclusive allocation of resources with other types of systems (for example the painting processes above). At any instant, the (limiting) processor/memory unit B is being shared among the processing of 100 job loads of data.

Of course, to realize this in practice, we would have to increase the number of input devices of resource A from 1 to 100. Alternatively, if we already had a large availability of input devices A, and we were not using such sharing, the processor/memory resource B would still be limiting, or at full capacity, at a system throughput capacity of 6 job loads per hour. But then, if we introduced the interleaved sharing of resource B among the subtasks of the task threads, as described above, suddenly the throughput capacity would increase 100-fold to 600 job loads per hour.

This type of sharing, and the spectacular increases in throughput capacity resulting from its introduction, is merely another example of interleaving the processing of the subtasks of the task threads for each unit of production. Here the processor processes a sequence of thread sections or subtasks, with the individual sections or subtasks coming from a variety of different threads. It is no different in principle from the example involving painting of automobiles above.

In practice, given the numbers above, somewhat less than an extra throughput of 594 job loads per hour will be achieved by the coordinated sharing of the processor/memory unit B

among the subtasks of the different threads. Some of the processing time of B will be devoted to coordinated sharing activities, such as scheduling which job load is to be worked on next, switching from one job load thread to another, dispatching, or starting up, the next job load thread, and so on 5.

It takes processor/memory time to decide which subtask will be selected next (scheduling time); it takes processor/memory time to switch from a subtask of one thread to a subtask from another (context switching time); it takes time to switch processor activity from the execution of coordination tasks to the processing of an actual subtask of a thread (dispatch time). Thus some processor/memory resource time must be diverted from job load processing to carrying out the coordinated sharing procedure. [Later we will see that the time per thread, consumed by system resources for administration of the sharing, is a climbing function of the sharing level.]

In practice, the reason such complex resource-sharing is used is often not so much to improve existing throughput capacity, but to maintain throughput capacity, while cutting down on the resources needed. In other words we often use such sharing procedures to save expensive resources—often, unfortunately, without adequate consideration either of the consequences in terms of risk to throughput capacity, or of the cost of running the sharing procedure.

Definition of coordinated sharing of a resource

From the examples above, as well as the ideas about systems discussed so far, we can now give a concise definition of coordinated, inclusive sharing of a resource:

> *Resources R of a system are being shared in a coordinated, inclusive manner if, and only if, at any instant, more than one entity or entity set unit of system throughput is under processing by resources R, requiring that the resources R be engaged in (usually complex) coordination procedures for measurable periods of time.*

Alternatively in terms of threads of task activity:

> *Resources R of a system are being shared in a coordinated, inclusive manner if, and only if, R processes the individual threads (each thread of activity corresponding to a complete production task) in thread sections or subtasks, where processing of sections or subtasks from different threads is interleaved in time, requiring that the resources R be engaged in (usually complex) procedures for coordinating the processing of the interleaved threads for measurable periods of time.*

Thus a computer processor (or human) being is participating in coordinated sharing if it (he/she) has the processing of more than one job or process under way at any instant.

We now proceed to develop a fundamental relationship between throughput capacity (I), system resources (R), and the level of sharing s of the system resources R among threads of processing activities, or entities of throughput. To get at what is behind this relationship, we must distinguish strongly between when system resources are being under shared, when they are being fully shared, and when they are being over shared.

Throughput capacity and under sharing of system resources

Before we look at under sharing, we first define what we mean by sharing level. For example, with the car painting facility, when we had only one car being painted in each 30-minute sharing period, we say the sharing level is zero. If two cars are being painted at the same time within the sharing period, the sharing level is one. With 8 cars being painted at the same time we have a sharing level of 7.

The sharing period is also an important concept. It is the elapsed time (in some cases an average time) it takes to complete processing of a single thread of activity when there is no sharing. In the car painting example, the sharing period is 30-minutes, since it takes three 1-minute periods of car painting, and three 9-minute periods of coat drying, to paint a car. It is only by making fuller use of some of the time in this sharing period, perhaps by exploiting the paint drying time, that there is any opportunity for significant coordinated, inclusive resource sharing.

In the beginning of our sharing adventure, we might be timid, and share the resources only a little. With the car painting facility earlier, instead of a sharing level of 7, with one minute per coat, three coats per car, and 15 seconds resource-sharing administration time per coat, we might start with a sharing level of one, that is, only 2 cars being painted in the 30-minute sharing period. This would give a throughput capacity of 4 cars per hour. It would still take one minute per coat per car, but it would likely take only a little resource-sharing administration time per coat per car, since there are only 2 cars to keep track of—let us say 3 seconds per coat per car, or 9 seconds per car. In each 30-minute sharing period, total actual coat painting time would then be 6 minutes, and total resource-sharing administration time would be 18 seconds.

If we later increase the sharing level to 5, that is, 6 cars being painted in the 30-minute sharing period, throughput would now be 12 cars per hour. The resource-sharing administration time might now increase to 11 seconds per coat per car. In each 30-minute sharing period, total resource-sharing administration time would therefore be 198 seconds, or 3 minutes 18 seconds, and total painting time would now be 18 minutes, for as total of 21 minutes 18 seconds. We now do not have much idle time left, in the 30-minute sharing period, to get any benefit from increasing the sharing level further.

Finally, we could increase the resource sharing level to the point where throughput capacity is the greatest possible, that is, when the sharing level is 7, when we have 8 cars being painted at a time in a 30-minute sharing period. Throughput capacity is now 8 cars per 30-minute sharing period, or 16 cars per hour. We also assume, as before, that it now takes 15 seconds of resource-sharing administration time per coat per car. In each 30-minute sharing period, total painting time would now be 24 minutes, and total resource-sharing administration time would be 6 minutes, for a total of 30 minutes.

At this point, there are no further idle periods of time in each 30-minute sharing period, in which the painting resource could be shared among further cars. The painting resource is now completely occupied, either in putting a coat of paint on a car, or in administration of resource sharing.

At this point we say that the painting resource is fully shared, and that the throughput capacity increase due to resource sharing is at its maximum, giving us a the maximal or fully-shared throughput capacity. Prior to that, when the sharing level is less than 7, we can say that the painting resource is under shared.

Notice that resource-sharing administration time per coat increases as the level of sharing increases. We assert that it must do so, in general, since the more threads of activity there are to keep track of, at each decision point, the more time it will take per thread to administer the resource sharing among them.

Resource sharing administration time per car, or per thread of activity, must thus be a function of sharing level s. We denote this by the function a(s). In simple cases, a(s) will be a linear function of s, as in the painting example above, where we tacitly assumed that a(s) was 3 + 6s seconds per car.

Throughput capacity and over sharing of system resources

It might be thought that with the painting facility at its maximal or fully-shared throughput capacity of 16 cars per hour, or 8 cars being painted in each 30-minute sharing period, we cannot

increase the level of sharing any further, since the painting facility is fully occupied [2]. This is not so.

There is nothing to stop us from further increasing the level of painting resource sharing among the threads of activity, beyond the fully-shared level, to the over-shared level. We might have 31 cars being painted at a time, although we will not get any increase in throughput capacity from doing this; we will get a decrease instead.

For example, if we have a sharing level of 30 (that is, 31 cars), the painting facility could do the first coat on each of the 31 cars in turn, then the second coat on each of those 31, and then the third coat. But since there are now many more cars to keep track of, the resource-sharing administration time per coat will likely increase dramatically. If we use a(s) = 3 + 6s, as before, administration time will jump to 183 seconds per car, or about 3 minutes per car (or 61 seconds per coat per car). That means that the painting resource will be occupied with a single car for 3 + 3, or 6 minutes, so that throughput capacity per hour must be 60/6, or about 10 cars per hour.

This is clearly less than the peak, fully-shared, throughput capacity of 16 cars per hour, when the painting resource was being shared among only 8 cars at a time. System resources are being over shared.

The problem with resource over sharing is that as we devote more and more system resource time to sharing activities, since the resources are fully occupied, they must necessarily be doing less and less productive work—they are engaged in resource-sharing administrative activities. The greater the sharing level, the greater the time "wasted" in resource-sharing administration "overhead".

Equations relating throughput capacity and level of sharing with under shared resources

In the general case of sharing, we would expect the following. As we increase the level of sharing, the total resource-sharing time, during which resources are devoted to sharing coordination activities, will increase at an increasing rate. Initially, there will still be room for additional sharing of the system resources during the sharing period, and the system resources are under shared.

As we further increase the sharing level, throughput capacity I will continue to climb, eventually reaching its peak, or fully-shared, throughput capacity, at the point where the system resources are first fully occupied with both thread processing and resource-sharing administration. The system resources are now fully shared and there is no more room in the sharing period.

If the sharing level continues to increase, past the point where the resources are first fully occupied, then, because the resource sharing administration time continues to increase, the throughput capacity will fall, but in all likelihood at a shallow rate. The system resources are now over-shared.

If we increase the sharing level even further, throughput capacity will fall back to a very low level, in the neighborhood of the throughput capacity given by I = KR, when there was no sharing.

In other words, if you share the system resources, initially you get an improvement in throughput capacity, which can be improved further by increasing the sharing level further. But if you overdo it, and over share, you will get less than if you had shared the resources less, and you may even get very low throughput capacity, in which state the system is said to be thrashing. This is obviously an important relationship. We call it the resource-sharing relationship. However, since the system behaves in two separate ways, in one way when the resources are under shared, and in another way when the resources are over shared, it is impossible to formulate a single, accurate equation for throughput capacity versus sharing level that covers both behaviors.

We thus need two equations, one for the under-shared case and one for the over-shared case.

Derivation of the basic resource-sharing relationships

We now derive the basic resource-sharing equations, for both resource under sharing and over sharing. Readers may skip this more technical section, without loss of continuity.
Consider any system with throughput capacity I and resources R, and suppose that s is a sharing level less than the level that causes the resources to be fully shared.

Since s measures the number of additional threads being processed, in addition to the single thread when there was no sharing, and since we are free to increase s as much as we like, provided the sharing period is not fully utilized, this must effectively increase system resources R by sR. Hence, the throughput capacity must be:

$$I = KR[1 + s]$$

which is the simple *resource under-sharing equation*.

In the above resource under-sharing equation, a sharing level of s means that the resources are processing s + 1 threads at a time. Thus when s = 0, it means that is there is no resource sharing, and the equation reduces to I = KR, which we would expect.

In the painting example, when s was zero, there was no sharing, with one car being processed in a sharing period. Throughput capacity was 2 cars per hour, so that the following example of the under sharing equation, namely I = 2(1 + s), must apply.

If we allow s to increase to the level where the system resources are fully occupied, since there is now no further "free" time to exploit in the sharing period, then I becomes maximal.
Suppose S is the length of the sharing period, and suppose that p is the length of the (average, if necessary) processing time it takes to carry out the actual processing for each thread. [In the car painting example, p is the time, equal to 3 minutes, needed to put the 3 paint coats on a car.] Hence, at full sharing, since all the activity of the system resources, during S, will be consumed in both thread processing and resource-sharing administration, the number of threads under processing must be S/[p + a(s)].

Since the number of threads under processing is clearly also equal to 1 + s, at the full sharing point, or peak throughput capacity point, the following condition, called the full sharing condition, must hold at this point:

$$1 + s = S/[p + a(s)] \qquad \textit{full sharing condition}$$

If sharing administration time per thread is a(s) = 3 + 6s, as used in the car-painting example, we can solve this equation for s. The solution is s = 7, the correct answer, giving a peak throughput capacity of 2(1+ 7), or 16 cars per hour, as we deduced earlier.

If we increase s further, beyond this full-sharing point, as we saw with the painting example earlier, throughput capacity now behaves completely differently. It must fall gradually, since the resource-sharing administration time per thread keeps increasing, and there is only a fixed time available to the system resources for working on the subtasks of the threads of activity and resource-sharing administration. That means that we need a different equation relating I and s for the case where we want to consider throughput capacity beyond the peak, or full-sharing, throughput capacity.

Suppose a sharing level of s, where we have over sharing of the resources among the threads of activity (cars, in the car painting example). The time to complete the processing on all 1 + s threads under processing at any given instant must now exceed the sharing period S, since

that is the nature of over sharing. The total time needed to complete the processing of 1 + s threads is (1 + s)(p + a(s)), which exceeds S.

The number of threads that can be fully processed during a sharing period S must therefore be S/(p + a(s)).

Suppose now that T is the time unit by which throughput capacity is measured, for example, an hour, a day or a week, as in units per hour, units per day or units per week. The number of sharing periods in a time unit T must be T/S. Hence the number of threads processed per unit of time T, that is, the throughput capacity I when resources are over shared, must be given by:

$$I = [T/S][S/(p + a(s))] = T/(p + a(s))$$

If I = F is the peak or fully-shared throughput capacity, and s = f is the full-sharing level at which I = F, then F = KR(1 + f), and F = T/(p + a(f). Thus KR(1 + f) = T/(p + a(f). We can therefore eliminate T from the above equation for over-shared throughput capacity and rewrite it more usefully as:

$$I = RK(1 + f)[p + a(f)]/(p + a(s))$$

resource over-sharing equation

where R is the system resources, K is a constant, f is the sharing level at which throughput capacity peaks, p is the actual (possibly average) processing time per thread, a(s) is sharing administration time per thread, and s is the actual sharing level—and a sharing level that causes over sharing. When s is f, the level at which throughput capacity peaks, the above equation reduces to:

$$I = RK(1 + f)$$

as we would expect.

As an example of the application of the resource over-sharing equation, consider the painting system again, and suppose the level of sharing s is 60. We know that the system resources are fully shared when s = 7, so this is a clear case of over sharing. The over-sharing equation gives the throughput capacity as follows:

$$I = 2(1 + 7)[180 + 45]/[180 + 363]$$
$$= 6.7 \text{ per hour}$$

This number is well below the peak throughput capacity, as given by I = KR(1 + f) = 2(8) = 16 per hour.

Over sharing and thrashing

If the sharing-level s increases significantly beyond the level f at which throughput capacity I peaks, I will decline significantly, which is the phenomenon of severe over sharing or *thrashing* [1, 10].

When the system thrashes, a great deal of the time or activity of the system resources R is being consumed in resource-sharing administration, and throughput capacity is well below the maximal full-sharing level, even though the level of resource sharing is very, very high. Increasing the level of resource sharing further only results in even less throughput capacity, since the system resources R are already fully occupied.

It should now be clear that the increased throughput capacity caused by coordinated sharing of system resources always has to be "paid for" by a time-consuming, resource-sharing, administrative procedure. This is a rule that cannot be evaded. Readers who look around, will see it in operation in almost every agent-directed system, simple or complex, technological or social.

Resource sharing in a subsystem

If a resource-shared system is a subsystem in a greater system, it is useful to consider the sharing level as simply enhancing the subsystem's constant K, as in throughput capacity:

$$I = K[1 + s]R = kR$$

where k is a new constant. The overall system throughput capacity can then be deduced from the combination of subsystem throughput capacities.

3.0 An empirical equation for the under sharing and over sharing equations

Although it requires two separate equations to describe throughput capacity versus sharing fully, one for the case of under sharing and one for over sharing, it is possible to construct a mostly empirical equation that covers both cases to a fair approximation. We develop it as follows.

Consider any system with throughput capacity I and resources R. If we could introduce all the coordinated sharing possible for the system resources, without any cost in terms of resource-sharing coordination time, throughput capacity could be increased by some factor S, or (1 + S) fold.

This factor S could be as low as zero, if there is no potential for coordinated resource sharing among threads of activity. However, normally there will be some potential for improving throughput capacity by coordinated resource sharing. In the example of the paint shop earlier, S was 9 (when sharing administration time per coat was taken as zero), so that throughput capacity could in theory be increased by (1 +S) fold, or 10 fold. In the computer processor example, S was 99, so that throughput capacity could be increased 100 fold by resource sharing.

We call S the available sharing-enhancement potential (per harmonic resource unit of the system). S must always be zero or greater. It effectively increases system resources R by SR, when all possible sharing is being carried out, thus increasing R to R(1 + S). This is equivalent to saying that S, if made use of, will increase throughput capacity I by SKR, to I = KR(1 + S).
Suppose that s is the level of sharing actually undertaken, with s less than S. Here, as before, s measures the number of additional threads being processed, in addition to the single thread when there was no sharing. In the case of the painting example earlier, if s were 3, then the task threads for 4 cars would each be in some stage of being carried out by the painting resource at any one instant. Accordingly, the throughput capacity must be I = KR(1 + s), which is the simple version of the resource-sharing equation, as we have seen.

If we allow s to increase quite a bit, the number of threads of activity, or cars being painted in the painting example, will increase quite a bit too. But if s increases to the neighborhood of S, at some point all the activity of the painting resources will be consumed in either painting or resource-sharing administration. Throughput capacity I will have reached a peak.

Beyond this point, if we increase s further, as we saw earlier, throughput capacity now behaves completely differently. It must fall gradually, as the resource-sharing administration time keeps increasing, since there is now only a fixed time available to the system resources for working on both the subtasks of the threads of activity, and resource-sharing administration. That means that our equation I = KR(1 + s) is inadequate for the general case where we want to

consider throughput capacity beyond the peak. The basic resource-sharing equation must therefore be:

$$I = KR(1 + sF)$$

where F is a factor that comes strongly into play when the resource-sharing level s approaches and reaches the neighborhood of S. Hence F must be a function of s, or F(s), with the following empirical characteristics:

1. F(s) should be a reasonable measure of the fraction of system resource time available for generating throughput capacity, as opposed to being consumed by resource-sharing activities. It should thus be a reasonable measure of resources availability.

2. F(s) must have a value 1.0 when s is 0 and stay close to 1.0 when s is small, so that when the sharing level is well below S, the basic resource equation can reduce to I = KR(1 + s).

3. F(s) must decline from 1.0 as s increases a lot, at a sufficiently fast rate to force I to peak in the neighborhood of S.

Hence F(s) must be an empirical function of the form:

$$F(s) = (1 - e^{-z})$$

where: $z = k(S/s)^2$

This function F(s) satisfies the three conditions above, and peaks as needed at s = S, if the constant k is very close to 1.3. [Actually I is always at a maximum, in the function FR(1 + sF(s)) when z = 1.3.] The constants k and S can thus be used to adjust the function to suit any specific resource-sharing system, such as the car-painting system we have used as an example.

Accordingly, we conclude that the basic resource-sharing equation must be:

$$I = KR[1 + sF(s)]$$

where: $F(s) = (1 - e^{-z})$

and: $z = k(S/s)^2$

Other formulations for F(s) may also be possible, although the one given above works reasonably well in practice.

Suppose now we apply the sharing equation above to the example of the painting resource earlier. We have to get the constants right to model that system correctly. We therefore select a low value for k of 1.4, and use S = 9.

[When k is high, I will peak when sharing level s is greater than S, and when k is low, I will peak with s less than S. We need a low k if want to allow for, or model, the significant time spent by the painting resource in resource-sharing administration, which caused the resource-sharing level to peak at 7 (that is, threads of 8 cars being shared, and throughput of 16 per hour), instead of the theoretically possible s = S = 9 (threads of 10 cars being shared, and throughput of 20 cars per hour).]

When there is no sharing, and s = 0, and I = KR, we saw that throughput was two cars per hour, so KR must be 2. Hence the resource sharing equation for the painting example could be:

$$I = 2[1 + sF(s)]$$

where: $F(s) = (1 - e^{-z})$

and: $z = 1.3(7/s)^2$

If s = 0, the basic resource-sharing equation gives I = 2. If s = 2, it gives a throughput capacity I value of about 6, as we would expect. With s = 4 the equation gives an I of about 10, again what we could work out from the example. When s = 7, the s value at which throughput capacity should peak at 16, we get a somewhat greater departure from what we would like, namely a value for I of near to 13, which is about the best this empirical equation can do.

The two exact sharing equations, namely the under sharing and over sharing equations, are linear, and give an "angled" or "sharp" peak throughput capacity. The single empirical equation rounds out this sharp peak, necessarily giving a lower throughput capacity at the peak than is exactly correct This is the price that must be paid if a single equation is made to fit both the under and over sharing cases.

If s is a ridiculous 21, so that we are sharing well beyond the level that makes any sense, I now drops to about 8, according to the empirical resource-sharing equation, and the system is beginning to thrash.

4.0 Resource sharing versus increasing system resources

The complexity and time consumption of the sharing procedure will likely hint at when it might be better to pay the financial price of increasing system resources R, and forego using resource-sharing to increase throughput capacity. The simplest way to improve throughput capacity, although not necessarily the least expensive way, will always be to increase system resources R. The more complex way to increase throughput capacity, attractive to computer scientists, is to look for and introduce a coordinated, resource-sharing procedure. Merely increasing system resources may appear to some to be too limiting. Indeed, since resources are always scarce, as each new type of system appears, it can confidently be expected that research, particularly computer science research, to find a way to increase system throughput capacity, by resource-sharing, will not be far behind.

References

1. Anderson, T.E., Lazowska, E. D., and Levy, H. M., The performance implications of thread management alternatives for shared-memory multiprocessors, IEEE Trans. On Computers, 38(12), 1989, pp1631-1644.

2. Barrow, J.D. *Impossibility – The limits of science and the science of limits*, Oxford University Press., Oxford, 1998, p 146.

3. Bradley, J. A Systems-Performance Equation for Multi-thread Systems, University of Calgary, Department of Computer Science, Research Report No. 2000-656-08, April 10, 2000

4. Date C.J. *An Introduction to Database Systems*, 7th Edition, Addison-Wesley, Reading Mass., 2000.

5. Gray, J. and Reuter A. *Transaction processing, Concepts and Techniques*, Morgan Kaufmann, Palo Alto, California, 1993.

6. Hennessy, J. L., and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Palo Alto, California, 1990

7. Hillier, F. S. *Introduction to Operations Research*, McGraw-Hill, New York, 1995.

8. Laowska, E.D, Zahorjan, J., Graham, G.S., and Sevcik, K. C., *Quantitative System Performance,* Prentice-Hall, Englewood Cliffs, NJ, 1984.

9. Sauer, C. H., and Chandy, K. M., *Computer System Performance Modeling*, Prentice-Hall, Englewood-Cliffs, N.J., 1981.

10. Silberschatz, A, Peterson, J. L., *Operating Systems Concepts*, 6th Ed, Addison Wesley, Reading, Mass., 2002.

11. Weinberg, A.M. On the relation between information and energy systems, National Conference of the ACM, 1980, reproduced in "Maxwell's Demon", H.S. Leff and A.F. Rex, editors, Princeton University Press, 1990, p. 116.

12. Zurek, W. Thermodynamic cost of computation, algorithmic complexity, and the information metric, Nature, Vol. 342, 1989, p. 119.