Cake: a concurrent make CASE tool

Citation for published version (APA):

Document status and date:
Published: 01/01/1990

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Memorandum COSOR 90-25

Cake, a concurrent Make CASE tool

J. van Oorschot
A. Dekkers
Cake, a concurrent Make CASE tool

Jan van Oorschot (i)
Anton Dekkers (ii)

(i) Delft University of Technology
P.O.Box 5031
2600GA Delft
The Netherlands
e-mail: JPMvOorschot@et.tudelft.nl

(ii) Eindhoven University of Technology
P.O.Box 513
5600MB Eindhoven
The Netherlands
e-mail: wscoade@win.tue.nl
1 Abstract

Letting a user generate tasks in a multitasking computer environment will result in a serial stream of tasks, and thus a less than optimum use of the available computing and storage devices. When presented with a number of concurrent tasks, the multitasking operating system will be able to schedule the tasks in order to optimally fill the resources and so improve the throughput.

In a programming environment, serial execution is often initiated by the well-known Unix tool Make. The concurrent equivalent is the newly developed CASE tool Cake (Concurrent Make). This CASE tool is developed to describe a network of tasks and present the operating system with a flow of concurrent tasks. The overall response time of the network is less than the corresponding serial execution of the tasks.

Measurements in an OS/2 environment show that for a characteristic program development project, the overall throughput increases when offering the tasks in parallel instead of in sequence. By converting a standard Make description file in a Cake description file, the time needed to perform all tasks can be decreased by 25%. Three concurrent tasks are enough to almost optimally use the available resources in a single processor environment.

Modelling the Cake system using a simple queueing model results in a tool that can be used to predict the behaviour of the Cake performance when changing parameters such as the number and speed of the processing and storage devices. Comparison of the model results and measurements confirm that the model correctly predicts the behaviour of the system.

2 Introduction

Current applications of workstations as personal computer have generated a large number of single user, multitasking computer environments. The operating systems involved are VMS, UNIX and OS/2 operating on hardware like the VAX 2000, Sun SparcStation and IBM PS/2-70. A single user tends to use less than the full capacity of such a workstation. This is caused by the inability to generate a constant flow of tasks that the operating system (OS) can schedule to fully use the available resources like the processing and storage devices. A single user will typically present the OS one task, wait for the result, think about this and repeat the process.

This serialisation can also be found in simple CASE (Computer Aided Software Engineering) tools like the UNIX Make tool. A network of tasks is described as sequence of serially executable tasks. In this case, even though all tasks are known in advance to the OS, the resources are not used in the best possible way.

This paper describes a CASE tool that, given a description of a collection of tasks, provides the OS with as many concurrent tasks as possible. The OS should be able to schedule these tasks in such a way that the processing and storage devices are used optimally.

To evaluate the performance of the resulting system, and to predict the effect of adding resources, a simple queueing model of the system is described and the performance parameters are compared with measurements in an OS/2 environment.

Section 3 describes the principles involved in the concurrent execution of a system of potentially concurrent tasks. Section 4 describes a laboratory test of the Cake performance. Section 5 describes a simple queueing model modelling the system. Section 6 gives an comparison of the measurements and the model results. Section 7 does a "What-If" analysis of the system if an extra disk is added. In section 8 some concluding remarks are given.
3 Concurrent execution of independent tasks

The type of CASE tools involved (both Cake and Make) allow users to describe the sequence of actions as a dependency tree. Each node $O_{ij}$ in the tree represents a file system object (node $<i>$ on tree level $<j>$). The root of the tree $O_{11}$ represents the goal object, normally a file containing an executable binary. The collection of arcs $\{A_{ij}\}$ leaving a node represents a series of actions done on the target nodes, resulting in a newer version of the originator node.

The user can request the system to validate an object $O_{ij}$ in a given tree $T$. The system resolves this request using the following recursive algorithm:

```plaintext
validate($O_{ij}$)
{
    validate($O_{(i+1)k}$), $O_{ij}$ dependent on $O_{(i+1)k}$
    if $O_{ij}$ doesn't exist then { $O_{ij}$ is invalid } 
    for all $O_{(i+1)k}$, $O_{ij}$ dependent on $O_{(i+1)k}$
        if $O_{(i+1)k}$ newer then $O_{ij}$ then { $O_{ij}$ is invalid }
        if $O_{ij}$ is invalid
            execute actions $A_{ij}$ to validate $O_{ij}$
}
```

This recursive algorithm walks through the dependency tree and will execute all the actions needed to generate the requested valid object $O_{ij}$. If object $O_{ij}$ is already valid, no action will be taken.

3.1 The Make approach

The original Make utility implements a dependency tree like the one described in the previous section. Make implements the execution of the actions $A_{ij}$ needed to generate object $O_{ij}$ in a straightforward way:

- activate $A_{ij}$
- wait for $A_{ij}$ to terminate

This approach results in the sequential execution of the actions in the dependency tree. The order of execution will be a "depth-first-search" order.

3.2 The Cake approach

Although the dependency tree used by Cake closely resembles the one used by Make, the execution algorithm differs strongly. Apart from the dependency tree, two extra data structures are generated, a "todo-list" and an "action-list". The "todo-list" contains actions $A_{ij}$ that can be executed, the "action-list" contains actions $A_{kl}$ that are being executed. There is a maximum number of jobs $M$ that can be executed concurrently.

The following algorithm is used to update the "todo-list" and the "action-list":

{ initialisation }

- collect in "todo-list" all actions $A_{ij}$ belonging to invalid objects $O_{ij}$ that are not dependent on any other invalid node $O_{kl}$.
- move as many actions from the "todo-list" to the "action-list" as possible.
• wait for an action \( A_j \) to terminate
{ action \( A_j \) terminates }
• check if the parents of \( A_j \) need to be added to the "todo- list".
• move those actions \( A_j \) from the "todo-list" to the "action-list" that belong to the object \( O_{ij} \) that is the furthest removed from the root of the dependency tree, until there are \( M \) actions in the action-list or until the todo-list is empty.
• wait for an action \( A_j \) to terminate.

The scheme results in the concurrent execution of at most \( M \) tasks. It is possible that due to certain tree configurations and execution characteristics it isn't possible to execute the maximum number of jobs concurrently. The above described algorithm has a good worst-case behaviour and will in general execute as many jobs concurrently as possible.

4 Measurement
To determine the performance of the Cake system, measurements were performed using an artificial dependency tree. Both the objects \( O_{ij} \) and the actions \( A_j \) were known quantities in the measurement. The dependency tree is shown in Figure 1, the algorithm for the action \( A_j \) in Figure 2. The test environment consists of 9 jobs that can run concurrently, each using a determined amount of processing and disk capacity.

![Dependency Tree](image1)

![Action Flow](image2)

Figure 1: Dependency tree for measurement  
Figure 2: Flow for action \( A_j \)

The jobs have the following characteristics:

\[
K = 9 \\
N = 9 \\
w_d = 0.231 \text{ sec} \\
w_c = C_n * 0.019 \text{ sec} \\
T_{\text{init}} = 1.85 \text{ sec}
\]

number of disk/cpu cycles  
number of jobs in tree  
average disk service time  
average cpu service time  
\( (C_n = 5 ... 20) \)  
action startup time
Tests have been done to measure the total throughput time of the jobs with a different number of concurrent jobs during the execution of each tree. The measurements are done for $M=1,2,3,4,5$. Note that the case ($M=1$) gives results equal to the Make system. $T(M)$ is defined as the time needed to execute the whole tree given $M$ concurrent jobs.

The results of these measurements are given in Figure 3.

![Figure 3: Measurement results of test tree](image)

As a comparison, a real life measurement is given. In Figure 4, the throughput of the compilation of the Cake-system itself is given for a number of different concurrent jobs. The Cake system consists of 11 C source-code modules and uses a large SCHEME library during linking.

![Figure 4: Measurement results of the Cake system tree](image)

5 Cake queueing model

Aim of this section is to indicate the profit Cake will give on Make. Of course this gain depends on the dependency tree used. In some pathological cases Make will perform just as
well as Cake. Here we derive the profit for what we consider as an average case, therefore the percentages of Cake response time in comparison to Make response time will be rough estimates for the response time gain with Cake in real life.

The computer system we consider here is a multitasking machine with a single user. The degree of multi-tasking $M$ can be specified and will vary from 1 to 5. A value of $M$ above 5 will not be meaningful, because the performance of the system will decrease rapidly as a result of extra swapping between memory and disk. This number $M$ turns out to be one of the two key-parameters in the performance comparison.

In our model the system is built of two devices, a CP and a Disk. The Cake tree consists of a number of jobs, $N$. During the processing of these jobs there will be constantly $M$ jobs present in the system. An exception is the final stage of the Cake task, for the few final jobs the number of concurrent jobs may be less. The root job will always be processed individually.

One single job will visit both the CP and the disk $K$ times. This number $K$ is not very important for a performance comparison between Make and Cake. Therefore an average over all jobs can be taken. Each time a job will be served during an exponentially distributed time with different means at CP and Disk. The ratio between these two means is the second key-parameter. We will assume here that the Disk is the bottle-neck. In our research we varied the ratio between the workloads at the CP and Disk from 0.4 to 1.0. This covers all important practical cases. Note that a switch of bottle-neck station will lead to almost equal results. Only minor differences in the model will occur by the starting and ending of the complete Cake task.

A last assumption we make concerns the overhead of a job. Each job will have an overhead caused by the program load and system initialization. This overhead is taken fixed for each job and during this time the CP as well as the Disk will be occupied.

The performance characteristics of the model above can be calculated with the Mean Value Analysis [1]. This is a recursive scheme. It will give in a straightforward way the response time at the CP, $S_c$, and at the Disk, $S_d$, for one visit as a function of the number of concurrent jobs and the workloads at CP and Disk ($w_c$ resp. $w_d$).

$$S_c(M) = \frac{(M - 1) \cdot S_c(M - 1)}{S_c(M - 1) + S_d(M - 1)} + 1 \cdot w_c$$

$$S_d(M) = \frac{(M - 1) \cdot S_d(M - 1)}{S_c(M - 1) + S_d(M - 1)} + 1 \cdot w_d$$

$$S_d(1) = w_d$$

$$S_c(1) = w_c$$

In the complete Cake task the last job is executed alone, this will take $K \cdot (w_d + w_c)$ seconds.

There are $K \cdot [(N-1)/M]$ rounds with $M$ jobs (the square brackets indicate the integer part of the number in between), these will take $S_c(M) + S_d(M)$ seconds each.

The remaining jobs except for the last one will make $K$ rounds together, and the time for a round of one of these jobs is found in the recursive determination of $S_c(M)$ and $S_d(M)$.

The total response time of Cake to a system with maximal $M$ concurrent jobs is now found by summing the three parts mentioned above and adding two extra terms: $N$ times the overhead, which was a constant for each job, and an extra visit to the CP. This extra visit is to compensate the first round, which will last longer because the system is not in equilibrium. Equilibrium is assumed for the other rounds.
We are also interested in the maximal profit that can be made. This profit is made for a situation with deterministic workloads and an infinite number of concurrent jobs allowed. In this case the time needed will be $N$ times the overhead plus $NK$ times the workload at the Disk (the bottle-neck station) plus $K$ times the workload at the CP (because the last job is processed alone) plus once the workload at the CP for the first job in its first round.

6 Verification

To check the assumptions made above we compared our results with the measured performance of Cake. We tested a Cake task with 9 jobs, an overhead of 1.85 seconds per job, 9 rounds for each job, a workload at the Disk of 0.231 seconds and a range of workloads for the CP. The overhead is obtained by comparing the measured and modelled MAKE. We plotted for a different number of concurrent tasks the measured response time against the ratio between the CP and Disk workloads. It is obvious that our model underestimates the qualities of Cake.

In the figures below we have plotted the gain of Cake towards Make for our test example. Other examples have given approximately equal plots. On the horizontal axis is given the ratio between $w_e$ and $w_d$. On the vertical axis is given the percentage of the response time of Cake in comparison with Make (100%). Each line in the plot indicates the gain-curve for a specific number of concurrent jobs. This value is 1 (i.e. Make), 2, 3, 4, 5. One can see that a situation with equal workloads at CP and Disk can profit most from Cake, as expected. Another remark is that larger overhead will decrease the profit, because of its simultaneous possession of both resources.
7 Prediction

The model can be used to predict the effect of certain changes in the configuration such as the adding of a server. As an example the model is expanded with an extra disk and the model results are calculated.

The visiting jobs are presumed to distribute their disk workload equally over the two disks. The results are displayed in Figure 9. Figure 10 gives the same figures, but now for the model with one CPU and one disk.

![Figure 9: Model result with 1 CPU and 2 disks](image1)

![Figure 10: Model results with 1 CPU and 1 disk](image2)

The results show that the adding of a disk improves the throughput of the system, especially if the processes are disk-bound (20%). If the processes are CPU bound, the gain of adding an extra disk is small (8%).

This example shows that the model can be used to give a quantitative prediction of the effect of changing the computer configuration.

8 Conclusion

The Cake system is modelled using a simple queueing network model. The model contains two servers, a CPU server and a Disk server. The Cake tasks are modelled as N jobs, M at a time making K rounds through the queueing system. The servers are supposed to have an exponentially distributed service times.

The time gained by using the Cake system instead of the Make system is determined by the CPU/Disk ratio of the tasks to be performed. For example, if the average CPU/Disk ratio is 30%, the maximum speed improvement will be less than 30%. Theoretical maximum time gain using one CPU and one Disk is 50% for equal deterministic workloads.

Speeding up one of the resources will result in a change of the CPU/Disk ratio, and so in the time gain. The bottle-neck resource directly determines the system throughput.

A parallel execution mechanism such as found in the Cake system is very useful in optimizing the use of multiple resources in environments such as Local Area Networks (LAN’s) and transputer networks.

9 References


<table>
<thead>
<tr>
<th>Number</th>
<th>Month</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990-01</td>
<td>January</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the asymmetric shortest queue problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Part I: Theoretical analysis</td>
</tr>
<tr>
<td>1990-02</td>
<td>January</td>
<td>D.A. Overdijk</td>
<td>Meetkundige aspecten van de productie van kroonwielen</td>
</tr>
<tr>
<td>1990-03</td>
<td>February</td>
<td>I.J.B.F. Adan, J. Wessels, W.H.M. Zijm</td>
<td>Analysis of the asymmetric shortest queue problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Part II: Numerical analysis</td>
</tr>
<tr>
<td>1990-04</td>
<td>March</td>
<td>P. van der Laan, L.R. Verdooren</td>
<td>Statistical selection procedures for selecting the best variety</td>
</tr>
<tr>
<td>1990-05</td>
<td>March</td>
<td>W.H.M. Zijm, E.H.L.B. Nelissen</td>
<td>Scheduling a flexible machining centre</td>
</tr>
<tr>
<td>1990-06</td>
<td>March</td>
<td>G. Schuller, W.H.M. Zijm</td>
<td>The design of mechanizations: reliability, efficiency and flexibility</td>
</tr>
<tr>
<td>1990-07</td>
<td>March</td>
<td>W.H.M. Zijm</td>
<td>Capacity analysis of automatic transport systems in an assembly factory</td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>M 90-09</td>
<td>March</td>
<td>P.J.M. van Laarhoven</td>
<td>Production preparation and numerical control in PCB assembly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W.H.M. Zijm</td>
<td></td>
</tr>
<tr>
<td>M 90-10</td>
<td>March</td>
<td>F.A.W. Wester</td>
<td>A hierarchical planning system versus a schedule oriented planning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J. Wijngaard</td>
<td>system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W.H.M. Zijm</td>
<td></td>
</tr>
<tr>
<td>M 90-11</td>
<td>April</td>
<td>A. Dekkers</td>
<td>Local Area Networks</td>
</tr>
<tr>
<td>M 90-12</td>
<td>April</td>
<td>P. v.d. Laan</td>
<td>On subset selection from Logistic populations</td>
</tr>
<tr>
<td>M 90-13</td>
<td>April</td>
<td>P. v.d. Laan</td>
<td>De Van Dantzig Prijs</td>
</tr>
<tr>
<td>M 90-14</td>
<td>June</td>
<td>P. v.d. Laan</td>
<td>Beslissen met statistische selectiemethoden</td>
</tr>
<tr>
<td>M 90-15</td>
<td>June</td>
<td>F.W. Steutel</td>
<td>Some recent characterizations of the exponential and geometric</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>distributions</td>
</tr>
<tr>
<td>M 90-16</td>
<td>June</td>
<td>J. van Geldrop</td>
<td>Existence of general equilibria in infinite horizons with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C. Withagen</td>
<td>exhaustible resources. (the continuous time case)</td>
</tr>
<tr>
<td>M 90-17</td>
<td>June</td>
<td>P.C. Schuur</td>
<td>Simulated annealing as a tool to obtain new results in plane geometry</td>
</tr>
<tr>
<td>M 90-18</td>
<td>July</td>
<td>F.W. Steutel</td>
<td>Applications of probability in analysis</td>
</tr>
<tr>
<td>M 90-19</td>
<td>July</td>
<td>I.J.B.F. Adan</td>
<td>Analysis of the symmetric shortest queue problem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J. Wessels</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>W.H.M. Zijm</td>
<td></td>
</tr>
<tr>
<td>M 90-20</td>
<td>July</td>
<td>I.J.B.F. Adan</td>
<td>Analysis of the asymmetric shortest queue problem with threshold</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J. Wessels</td>
<td>jockeying</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W.H.M. Zijm</td>
<td></td>
</tr>
<tr>
<td>M 90-21</td>
<td>July</td>
<td>K. van Harm</td>
<td>On a characterization of the exponential distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F.W. Steutel</td>
<td></td>
</tr>
<tr>
<td>M 90-22</td>
<td>July</td>
<td>A. Dekkers</td>
<td>Performance analysis of a volume shadowing model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>J. van der Wal</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Month</td>
<td>Author</td>
<td>Title</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>M 90-23</td>
<td>July</td>
<td>A. Dekkers J. van der Wal</td>
<td>Mean value analysis of priority stations without preemption</td>
</tr>
<tr>
<td>M 90-24</td>
<td>July</td>
<td>D.A. Overdijk</td>
<td>Benadering van de kroonwielflank met behulp van regeloppervlakken in kroonwieloverbrengingen met grote overbrengverhouding</td>
</tr>
<tr>
<td>M 90-25</td>
<td>July</td>
<td>J. van Oorschot A. Dekkers</td>
<td>Cake, a concurrent Make CASE tool</td>
</tr>
</tbody>
</table>