Introduction to the CAFE scheduling and simulation testbed

Asbjoern M. Bonvik

July 10, 1995

1 Introduction

The purpose of the CAFE scheduling and simulation testbed is to serve as a factory floor dispatcher integrated in CAFE, and also to permit exploration of different operating strategies in semiconductor manufacturing. It is able to identify most important characteristics of the fab state by reading the CAFE database. This includes the repair state of the machines, the locations and progress status of the lots, reservations made on the machines, etc. So far, the scheduler does not consider staff constraints, because these are not well represented in CAFE.

The simulator is compiled from the same code as the scheduler. It takes the same initial state of the fab and projects it into the future, using the scheduling algorithm chosen. The difference is that the simulator will generate lot releases and random events such as machine failures and repairs. This allows comparison of different scheduling approaches.

This document is meant to be an introduction to the testbed. A more detailed user's guide may appear later. There will undoubtedly be remaining questions, especially for those that need to modify parts of the testbed, but no comprehensive programmer's guide is planned. The actual code is always the definitive documentation of the program for modification purposes, and it is written with clarity and modularity as important design objectives.

A few words about the typographical conventions of this document: a `typewriter` style is used for computer output examples, input examples that must be typed exactly as stated, variable names, etc. An *italic* type is used

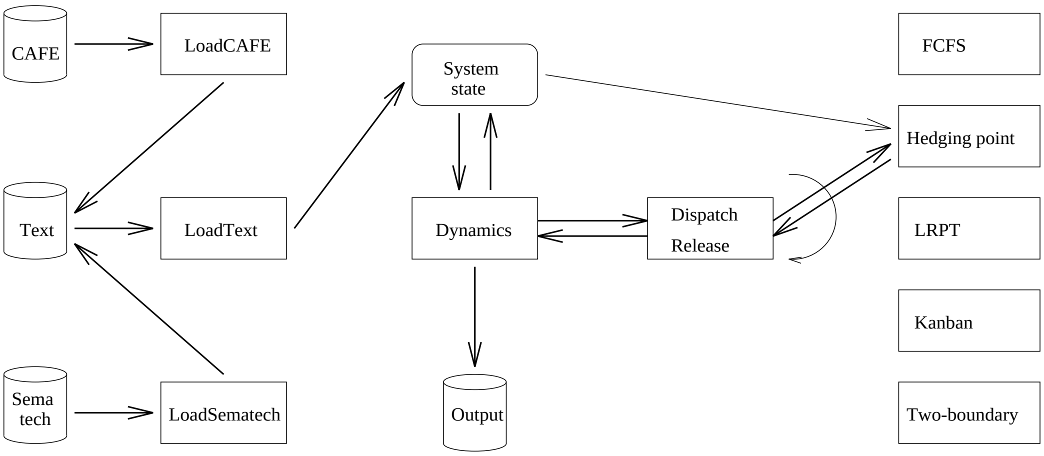

Figure 1: The overall program organization

in the place of some actual value in input examples, and very occasionally in the usual context of emphasizing new concepts.

2 Design

The testbed design is based on [3] and work done by Sungsu Ahn [1]. It can be divided into three logical modules, the front end, the system dynamics, and the control. These modules communicate through a common perceived system state. The front end loads an initial state, the dynamics evolve the state as the schedule is computed, and the control module decides which actions are taken based on an inspection of the system state. This separation can be seen in figure 1. There are three distinct front ends, and there can be any number of different control modules to be selected by the user. So far, four distinct control modules are implemented in the testbed.

The testbed does not change the CAFE database in any way. It just reads information from the database and comes up with a schedule recommendation. Schedule implementation, such as making reservations, releasing lots, and doing operations must be done by the user. Neither does the testbed assume that its recommendations are followed to the letter. The next time the scheduler is executed, it will look up what actually has been done by rereading the database, and recommend a schedule from there.

The real time scheduler and the simulator are compiled from exactly the same code. It is also possible to compile "lightweight" executables that do not include the CAFE front end with the Gestalt and Ingres code required by CAFE. This allows text-only versions to be distributed to sites that do not have a CAFE installation.

2.1 Input

2.1.1 The text front end

When it is not necessary or suitable to retrieve data from CAFE, the text interface can be used. This interface is a plain text file in free format (any number of spaces, tabs, or newlines between fields). There are two files to be read in, referred to as the resource and cache file, respectively. The resource file contains static data such as opening hours, machine failure distribution, process demand and release data, and process descriptions. The cache file is an image of the current fab state, and contains the state of each machine and lot in the fab.

An example of a resource file is given in Appendix A, and an example of a cache file is given in Appendix B. If no names for these files are given on the command line, they will default to *facility*`.src` and *facility*`.sch`, respectively. Here, *facility* represents the name of the facility to be scheduled, either taken from the environment variable[1] `facility` or from the testbed command line (`-f` option). Both files are assumed to be in the current working directory. Keywords in the files, such as `HOURS` or `PROCESS` are required to be in uppercase.

In the resource file, there is an estimated time to fail and time to repair distribution for each machine; and for each process, there is a distribution of the times between lot releases. These distributions are given as one of:

`nil`		
`constant`	*interval*	
`exponential`	*rate*	
`erlang`	*shape*	*scale*
`gamma`	*shape*	*scale*
`weibull`	*shape*	*scale*

[1] Usually, environment variables under UNIX are all uppercase, but not this one.

`beta`	*alpha*	*beta*
`normal`	*mean*	*sigma*
`uniform`	*low*	*high*

The distribution type keywords are not case sensitive, so typing `Weibull` means the same as `weibull`. The time unit used in these distributions is seconds, so the lot release rates etc can be quite small numbers. Notice that the exponential distribution has mean *1/rate*, while the closely related Erlang and gamma distributions have means *shape * scale*. Entering the distribution name `nil` without any parameters will cause the distribution not to be sampled. This can be used to state that a machine is perfectly reliable and thus never will fail.

2.1.2 CAFE front end

The CAFE front end is the only part of the testbed that uses CAFE/Gestalt data types. It accesses the CAFE database to figure out the system state, and communicates with the rest of the testbed by writing files that are read back in through the text front end. This simplifies program maintenance, since objects used in the testbed are created in one module only. Normally, only the cache file is rewritten during a run of the scheduler from CAFE (using `-C` command line option), but using the command line option `-R` will force the resource file to be rewritten as well.

The CAFE front end will estimate the repair and failure distributions from the machine logs. This procedure assumes that the actual distribution can be approximated by a gamma distribution, and uses the method of moments [8] to fit the parameters of the gamma to give the same mean and variance as the observed data. Machines with insufficient data are given a `nil` distribution. The number of observations required to print an estimate is controlled by the `-z` command line option.

The testbed can only represent one facility at a time, and inter-facility lot moves can not yet be represented. The facility to be scheduled is set as an environment variable by CAFE. When running the testbed from outside a CAFE session (e.g, as a batch run), it is necessary to either set this environment variable from the shell (e.g., `setenv facility ICL`) or to use the `-f` command line option.

2.1.3 Sematech front end

Sematech has developed a format for describing the products and facilities of a semicoductor fab [4]. This format is a set of ASCII text files with a rather dense data representation. The Sematech front end of the testbed can read these datasets and convert them to its own text format. This front end is included in all versions of the program (scheduler and simulator, CAFE and lightweight).

The Sematech data do not include a fab state, so any schedule developed from such a set starts with the fab empty (no work in progress) and all machines up. It is necessary to have a long initial simulated period to approach steady state in the fab. The Sematech data include information about operators, scrap, and rework that is currently not used by the testbed.

The Sematech front end is used by giving the `-S` command line option followed by the name of the dataset. For example, type

```
schedsim -S set1 -a HP -h 365
```

to simulate the fab in Sematech dataset 1 for a year. This command line option implies `-f set1`, `-c set1.sch`, and `-r set1.src`.

2.2 System dynamics

2.2.1 Simulation paradigm

The fundamental design of the real time scheduler is that of a discrete event simulation or simulation scheduler [2]. The current system state is evolved into the future, and decisions are made whenever necessary. These decisions can be invalidated by later simulated events, so only the the actions finally committed to are written to the schedule. This allows for the high degree of commonality between the scheduler and simulator.

Simple control modules can be written as dispatch rules that only consider the state of the system at the time they are invoked, but this design does not prevent more sophisticated algorithms from predicting future states of the system and acting on that. Such algorithms will need their own internal representation of the state, since no control module is allowed to modify the common system state.

2.2.2 Event types

There are four event types that are considered by the scheduler. These are

LOT ARRIVAL	Generates a DECISION if machine is free.
OPERATION START	Generates a OPERATION END, writes a schedule entry.
OPERATION END	Generates a DECISION at this machine, a LOT ARRIVAL at the next machines for lot.
DECISION	Triggers a call to the control module, may generate an OPERATION START.

The simulator has the additional events

FAILURE	Interrupts operation, generates a REPAIR.
REPAIR	Restarts operation, generates DECISION if idle, generates the next FAILURE of the machine.
LOT RELEASE	Generates LOT ARRIVAL at first machines.

The hedging point control policy will generate lot releases also when executed as a scheduler.

2.2.3 Setups, operations, and transfers

The testbed can represent operation durations on a per-lot basis, two kinds of setup times, and a transfer time between any two successive operations. For now, all this information is stored in the process description, but this may be changed in the future. For every operation in the process description there has to be a `DURATION` field that describes the length of the operation in seconds per lot. It is assumed that the operation requires exactly one of the machines listed in the `MACHINE` field, and that the machine will be occupied for the entire duration of the operation.

Operations can have two kinds of setup behavior. A flow setup occurs every time a machine changes between two operations that are not exactly the same. For this purpose, two operations are considered equal if they are the same step in the same process. Note that this is a more narrow definition than in CAFE, where processflows for individual operations may be shared between processes. An example of this type of setup is adjusting the energy level of an implanter. Each operation may also have a group setup. The setup group the operation belongs to is defined by a text string, and is followed by a setup duration associated with changeovers between different setup groups. The group setup occurs whenever a machine changes between two operations with different setup groups. Operations without any explicit setup group are considered to belong to the same group, which is considered to be different from all named setup groups. An example of a group setup is changing the ion species of an implanter.

Every operation may also have a transfer time that will elapse before the lot will appear at the next machine. Examples of these data fields can be seen in Appendix A.

2.3 System control

2.3.1 The dynamics - control interface

To make control modules interchangeable, it is necessary to define a clear interface between the system dynamics and the control module. This interface has the form of the machine asking the current control module "What do I do next?" The answer will be of the form "Load lot X at time Y" or simply "Nothing".

Each control module must have a single entry point that corresponds to this question and response. This entry point is a function that takes a pointer to a machine structure and a pointer to a `time_t` (a predefined type in UNIX) as arguments and returns a pointer to a lot structure. The machine pointer is set by the caller to point to the machine querying, and the control module will set the value of the `time_t` to the time the lot is to be loaded. The machine will assume that the lot returned is in fact ready to be loaded on the machine. If no suitable lot is found, the control module will return NULL. The loading time variable is then unchanged.

The testbed has a table of pointers to the control module entry points that are implemented. This can be used to select control module prior to running the program, and conceivably also to change control module dependent on run-time conditions. E.g., it might be desirable to switch dispatch rules based on the overall system load, or have different machines be scheduled by different rules. The current version of the program has not been prepared for run-time changes of control module.

2.3.2 Common aspects

The control modules implemented have some common features. They all understand lot priorities, opsets, and reservations. A high priority lot will only queue behind other high priority lots that are ranked before it by the current scheduling policy. Normal priority lots will only be served when all high priority lots have been served. Similarly, normal priority lots always precede low priority lots. The priority scheme is not preemptive, so a low priority lot that has started an operation is allowed to complete, even if a high priority lot arrives at the machine.

If a lot queues for an operation that is included in an opset, and where one of the succeding operations in the opset cannot be done because all machines that can be used for that task are down, the lot will not be loaded. Other lots that do not need to use the machine that is down will then be loaded instead. This ensures that opsets are completed within reasonable time of their start.

None of the control modules attempt to optimize the setup sequence at the machines. This is still a research topic, and various heuristics will be implemented in later versions of the testbed.

The loading time computed is at the start of the first time window (between reservations, lab closings, and other jobs) that is large enough to accomodate the task. If an operation has a duration longer than the lab opening hours on any day of the week, a warning message will be generated, and the lot will be scheduled for immediate loading. This will violate the lab opening hours, but reservations on the machine will still be respected.

In all of the rules, if the primary ordering criterion yields a tie, the lots that have waited longest are served first.

2.3.3 First come, first serve approach (FCFS)

This corresponds to a fair queuing rule at every machine, where lots arriving earlier are served before later arrivals. Suitable for a lightly loaded research fab, but also a useful baseline for evaluation of other scheduling policies.

2.3.4 Fewest remaining operations (FRO)

This rule prioritizes lots that have few steps remaining in their process descriptions in order to get lots out of the fab as quickly as possible. Suitable for a lightly loaded fab where the mean lead time is an important performance measure. See [7] for details. This policy is known to be stable.

2.3.5 Least remaining processing time (LRPT)

This policy is similar to the FRO policy, but orders lots according to the sum of processing times at all remaining operations. Usage as the FRO policy, but may be slightly better if there are very large variations in the time required for operations.

2.3.6 Hedging point policy (HP)

This policy attempts to track the demand for each product type as closely as possible, and adds in a *hedging point* of production surplus at each operation as described in [5]. The policy will then try to stay so much ahead of overall demand at that operation. The hedging points must be non-increasing along a process. The policy is implemented as a local dispatch rule at each machine, evaluating the pieces currently in the buffer. This policy can decide to wait for a while before loading a part to avoid building excessive inventory. It will also recommend release of new lots to track product demand. This policy is a close relative of the basestock policies of classical inventory control, and is suitable for a high volume fab with relatively few product types. The hedging points and surpluses given in the resource file are measured in wafers, demand in wafers per second. The hedging points need to be supplied by using the `HEDGINGPOINT` keyword in the resource file, and a parameter that is interpreted as the number of wafers this station should be ahead of demand. The hedging points along a process must be non-increasing, since the downstream operations can only process material that has gone through the upstream stages.

2.3.7 Two-boundary policy (HP2)

This policy tries to track overall demand as in the previous case, but adds constraints on the maximal allowable build-up of material in any single buffer. It can be seen as a combination of the hedging point / basestock policy of the preceding section and a kanban policy, which limits the local accumulation of material. The policy defines a cell consisting of a machine and the immediate downstream buffer in each process description. The machine cannot load a lot unless the number of lots in the cell is less than the limit value. Note that this blocks the machine from loading one particular operation, lots from another process or at another stage in the same process can be loaded, if their inventory limits are not reached. This policy has very attractive properties, see [11] or [10]. The policy parameters are specified by using the `HEDGINGPOINT` and `KANBAN` keywords in the resource file. The hedging point is interpreted as before, and the kanban number (an integer) is interpreted as the maximal number of lots in this cell.

2.4 Output

The scheduler can be run either in an interactive or in a batch mode. This section describes the interactive mode. See section 3.1 for a description of the batch mode.

2.4.1 Data loading messages

When loading from CAFE, the testbed will report any inconsistencies and other oddities it finds in the database. Examples of such messages are

```
TIMEAVAILABLE object ( , 0, 10:30:00, 11:00:00)
WARNING: Assumed that TIMEAVAILABLE objects mean lab *CLOSED*

WARNING: Found timeavailable object with NULL lower value,
assumed 00:00:00.

WARNING: Machine ICL_tubeB2, time 09/20/89 20:50:41,
bogus logentry up, skipped.

WARNING: Lot pflow1: active opinst has empty task list, lot skipped.

Note: Lot morse1: no active task, no ready task, please mark as DONE.

WARNING: Lot dgate: Task FURNACE-OPERATION has no machines.

WARNING: Lot dgate: Task DEPOSIT-MATERIAL has no duration, set to zero.

WARNING: Lot DA54 has NULL processinstance.

WARNING: No priority given for lot seles#1, set to NORMAL.
```

```
WARNING: Lot DA49: more than one ready task, only first one used.

WARNING: Lot tsai1 differs from process description in task
INSPECT-THICKNESS (#1 from end), operation name SINTER-WAFERS.

Note: Lot tsai1 put on HOLD due to process discrepancy.

WARNING: Lot seles#1 has no process description.

Note: Lot seles#1 put on HOLD due to process discrepancy.

WARNING: Lot NSOI2 differs from process description in task
INSPECT-THICKNESS, processflow EID 1778 (#1 from end),
operation name INSPECT-THICKNESS, EID 1960.

Note: Lot NSOI2 put on HOLD due to process discrepancy.
```

As the examples above show, the scheduler is quite picky when digesting data. It requires the tree of process flows emanating from the process description of the lot to match the task tree, so any creativity in inserting last minute tasks will probably cause the scheduler to disregard the lot. Any lots with multiple ready tasks or more than one top level task will also cause problems in this phase. The testbed also checks the consistency of cache and resource files when loading text input. The last six messages in the example above were generated by this front end.

2.4.2 The fab state presentation

After digesting the input, the testbed offers to present the starting state of the fab. If this is accepted by the user, the testbed will give a list of the machines and their states and input queues, and a list of the lots and their state. The lot state description will be tailored to fit the scheduling policy currently in use. An example is shown in Appendix C, in this case for the least remaining processing time dispatch rule.

A similar presentation will be offered at the end of the scheduled period.

2.4.3 The schedule

During schedule computation, some informational messages may occur, e.g., reporting that a lot has been put on hold for some reason. The schedule itself will be presented in a separate `emacs` window, and is also written to a file named `schedule` in the current directory. Command line option `-o` overrides this name of the schedule file. A typical schedule entry looks like this:

```
(:OPSET 12455 :TASK 12456 :MACHINE 5 :TIME ("09/06/94" "09:00:00"))
; t 778856400 Tue Sep  6 09:00:00 1994
; l dgate  p DGATE_2  o photo 45  t HMDS 101 x 0.00
; m ICL_HMDS  d 2700
```

The first line of the entry is meant to be read by LISP programs that present this to the user in a more friendly form. The information is given as entity id numbers that make sense inside CAFE. The next three lines are meant to be slightly more human-readable. The second line gives the loading time in terms of a UNIX `time_t` and a more civilized format, and the third and fourth lines state that, in this case, lot dgate, which has process description DGATE, version 2, is to be loaded on the machine named HDMS in ICL for the task HMDS (EID 101) in the opset photo (EID 45). This task is scheduled to take 2700 seconds. For hedging point calculations, it also reports that the current surplus at this operation is 0.0.

To simplify user modification of the schedule, the next few tasks beyond the end of the schedule horizon for each lot are also listed. The command line parameter opset depth (`-x`) controls the minimum number of opsets that are listed for each lot. These entries do not include any scheduled time for execution. Such extra entries can look like this:

```
(:OPSET 23020 :TASK 23023 :MACHINE 15)
; l nadir1  o dsro430  t INSPECT-THICKNESS  m ICL_ellipsometer  d 900
(:TASK 22632 :MACHINE 90)
; l wong1  t RCA-IONIC-CLEAN  m ICL_rca  d 0
```

An application that takes this schedule as input and presents it in an user-friendly way for acceptance or modification is planned. The schedule accepted by the user will then be passed on to another application, currently under development, that will make the necessary reservations in the database.

2.4.4 Performance measures

Both the scheduler and the simulator will compute performance measures. The testbed will offer to present these after the final state display. If accepted, it presents utilization, down time, and average queue length for every machine, followed by the lead times in weeks for each process where one or more lots departed the system within the run, and a breakdown of the time spent by each lot. See Appendix D for examples. A one-year simulation of the example in the appendices under the hedging point policy took less than one second of computer time on a Sun Sparcstation, running a non-CAFE simulator executable (`lschedsim`).

3 Using the testbed

3.1 Interactive and batch runs

There are three ways of running the scheduler; from CAFE, from the command line, and batch. The CAFE interface is described in section 3.3. When running in batch mode, all user interaction is turned off, and all output options are turned on. This can be useful for automated overnight runs or similar purposes. Batch mode is enabled by the `-b` switch on the command line.

3.2 Command line options

The testbed has these command line options:

`-a` *algorithm*	Chooses scheduling algorithm. Default: `FCFS`, other choices `HP`, `FRO`, `LRPT`
`-b`	Turns on batch mode. Default: Off.
`-C`	Reads fab state from CAFE, writes cache file. Default: Reads from cache file.
`-c` *filename*	Use named file as cache file. Default: *facility*`.sch`
`-d` *mm/dd/ss*	Sets the starting date of the schedule. Assumes input state to be valid at that time. Default: Date of running program.
`-f` *facility*	sets the facility to be scheduled. Default from `facility` environment variable.
`-h` *n*	Sets schedule horizon to *n* days. Default: 7

`-n`	No scheduling, parse input only. Default: Off.
`-o` *filename*	Write output schedule to named file. Default: schedule
`-R`	Reads resource data from CAFE, rewrites resource file. Implies `-C`. Default: Reads from resource file.
`-r` *filename*	Use named file as resource file. Default: *facility*`.src`
`-S` *set*	Use Sematech data set *set* at input. Implies `-f` *set* `-c` *set*`.sch` `-r` *set*`.src`. Will convert the dataset, write resource and cache file.
`-s` *seed*	Set the random number generator seed. Default: Taken from system clock, different for each run.
`-t` *hh:mm:ss*	Sets the starting time of the schedule. Assumes input state to be valid at that time. Default: Time of running program.
`-v` *n*	Sets output verbosity level. Default: 1
`-x` *n*	Sets opset depth to *n*. Default: 5
`-z` *n*	Sets the minimum sample size to estimate machine up and downtime distributions. Default: 10

A space may be entered, but is not required, between the option letter and its argument, if any. On the lightweight executables without CAFE interface, the `-c` option is implied if an argument is given on the command line. That is, the command `lsched myfab.sch` is equivalent to `scheduler -c myfab.sch`. Some common command line idioms:

```
nohup nice scheduler -c fab.sch -r fab.src -R -b >&! fab.log &
```

Make a batch run of the scheduler, retrieve data from CAFE and create new cache and resource files named `fab.sch` and `fab.src`, respectively. Store diagnostic messages in the file `fab.log`.

```
lsched -r fab.src -h 5 -a LRPT fab.sch
```

Use the lightweight (non-CAFE) scheduler to generate a 5-day schedule from the cached files, using the least remaining processing time dispatch strategy. Interactive mode, any messages will be written to the terminal.

```
lschedsim -r fab.src -h 700 -a LRPT fab.sch
```

Estimate consequences of that policy by simulating it for 700 days, assuming lot release distributions and machine failure/repair distributions to be stored in the resource file. Interactive mode again, since the simulation is also quite fast.

3.3 The user interface schfront

To simplify interactive use of the testbed, a Fabform interface has been written. This is the interface that is used when launching the testbed from the CAFE Scheduling menu. This form has eight fields. The first one is a choice between schedule and simulate, and the data source is a choice between CAFE, cache, and text file. If text file is chosen, enter the name of the file in the field for that. The other fields are hopefully self-explanatory, and a short message is shown on the bottom of the screen for further explanation as the fields are selected.

Press Control-X Control-C when the form is filled in, and answer y to the questions "really quit (y/n)?" and "Save new entries (y/n)?". These queries are less than informative, but are mandated by the Fabform user interface, and are not controlled by the application.

3.4 The program present_schedule

Sometimes it is desirable to compare the scheduled production with demand, especially in a volume fab. The program `present_schedule` reads a schedule as written by the scheduler and extracts initial surplus and demand data from the resource file. Usage:

```
present_schedule [-r resource][-o schedule] -p process -n k
```

This will result in a plot in a separate window, where cumulative production of the process at operation number `k` from the start is compared with cumulative demand. For example,

```
present_schedule -r test.src -p AMB-TEST_6 -n 6
```

The program will prompt for process name and / or operation number if the `-p` and `-n` command line options are omitted. The program uses x11giraphe3 [6] for displaying the plot. Temporary data are written to the files `temp.grp`, `temp.gdf`, and `temp_demand.gdf` in the current directory. These files are not removed after use, in case the user wants to reuse them for some purpose, but will be overwritten the next time `present_schedule` is run. Figure 2 shows an example of this plot format, generated by the command line just shown. The time axis is measured in seconds since 00:00:00 01/01/70, the internal units of the scheduler (and UNIX).

This program will probably evolve in response to user requests.

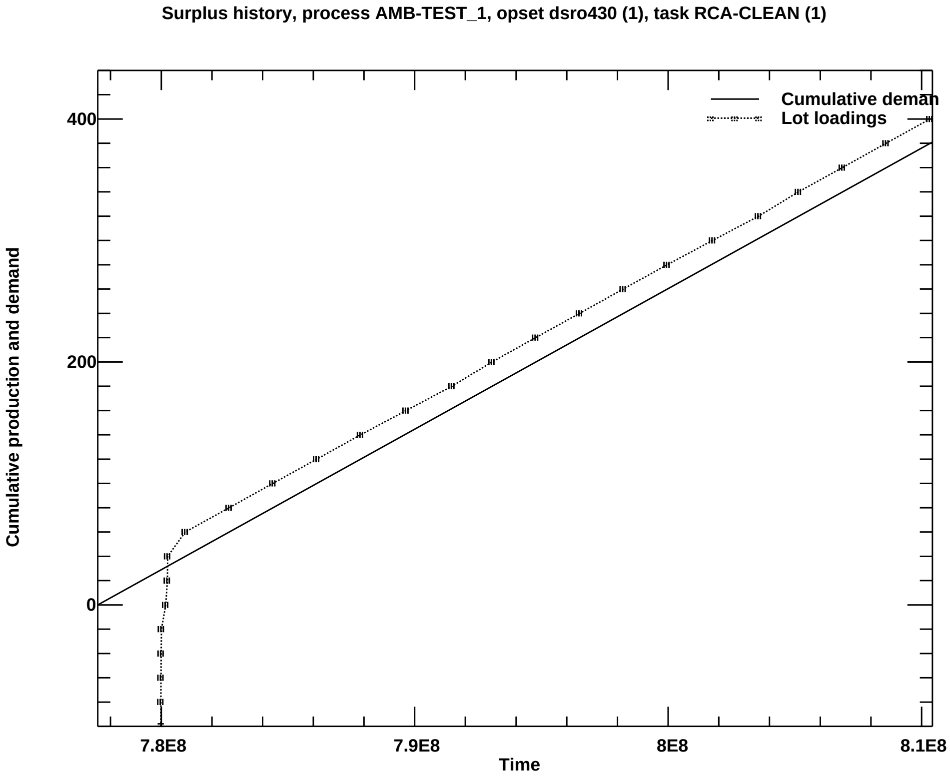

Figure 2: Example plot from `present_schedule`

4 Extending the testbed

The testbed is written to be modular and extendible. Two obvious ways to extend it are to add new control modules and new front ends.

4.1 Adding scheduling algorithms

Any rule can be added to the testbed, as long as the control can be expressed by using the interface described in section 2.3.1. The control module may perform any computation it wants, based on the system state and any additional information from the resource file. It may not modify the system state directly, but should keep its own state representation if it is necessary to project future system states.

It may be necessary to modify other modules and data structures to support new control modules. For example, a hierarchical policy needs hedging points per process, operation, and level of the hierarchy. This must then be entered and stored somewhere until the control module can use it. Another planned extension is a kanban policy that controls each process opset as a separarate kanban cell. This will require:

- a new control module
- a change to the opset data structure to store the number of kanban per cell
- a change to the text front end and the input format to enter these data
- a change to the dynamics module to keep track of the lots in each kanban cell
- another change to the dynamics module to ensure correct blocking and unblocking of machines, including a mechanism to trigger the decision module whenever a machine becomes unblocked
- a change to the main program to add the `-a kanban` switch to activate the policy

4.2 Adding front ends

It may also be desirable to load data from other sources. Some candidates are a direct link to the PFR and MFR, and using text formats defined by others. This can be done by adding new front ends that read the files in question and build the internal testbed representation of the system or that write text files to be read by the existing text front end. Such programs can also be built as stand-alone filters that are not linked to the testbed.

4.3 Adding lot release and routing policies

Some types of decisions are not yet represented in detail. One is the handling of lot releases. Unless the hedging point policy is used, only the simulator considers these, and it does so by handling lot releases as an external random process. There should probably be a choice for release strategy to be used in the same way as the scheduling algorithm is chosen. Two choices are the external random process currently implemented and a rule that adds another lot as soon as a lot departs the fab. This remains to be worked out.

It may also be desirable to control the routing decisions in the fab. In the current version, the lots queue in front of all machines that can do the next operation, and the machines choose which lots to load. A routing algorithm might be to queue the lot in front of the machine that has the shortest queue or has been idle longest. This is also left for a future version of the testbed.

4.4 Known bugs

A fundamental assumption is that the database gives a correct picture of the system state at the time a schedule is requested. This may not always be the case, but the program has no way of finding this out. A way to always have a correct representation is to wire all machines in the lab to detect failures themselves, automatically transmitting data to CAFE, and to use similar means of automatic data entry to detect when operations start and finish. This has so far been outside the scope of the project. If such automatic data collection is implemented, it would also be possible to monitor the transaction stream from the lab to invoke the scheduler automatically when certain changes such as machine failures or repairs occur. Currently, the state changes must be entered by the lab staff, so it is not unreasonable to require them to rerun the scheduler when it seems necessary to do so. See [9] for a different approach to real time scheduling of a semiconductor fab.

The testbed only looks at the nominal duration of an operation when fitting it in between reservations, fab closings, etc. If a setup is required, this may violate reservations and / or fab closing hours. This will be fixed as setup-savvy control modules are added for the next version of the testbed.

The testbed does not understand lot splits. A lot is assumed to stay together as it goes through the process description as stated in the database. If something of this kind is detected, the lot will be "set aside" for manual scheduling.

The testbed is not able to deduce from CAFE in which facility a lot currently resides, so if a lot can do the next operation on machines from several facilities, it will be considered for loading when scheduling any of these facilities. This can result in a lot being scheduled for processing in more than one facility, perhaps even for the same operation.

All machines are assumed to be in a neutral setup state at the start of a scheduling period or simulation. If the first operation to be done on a machine has a setup group specified, the associated setup time will be incurred. Similarly, the flow setup time of the first operation will always be incurred. This stems from the fact that neither CAFE nor the Sematech data format specifies a current setup state for the machines.[2]

There is no sure-fire way of identifying the fact that a labuser has already made a reservation to perform a certain task on a lot. The purpose slot of a reservation is a free-format 20 character text string, and there does not seem to be a clear standard for how to fill in this field. Therefore, if a lot has a ready task and the lot owner has signed up for future processing, the scheduler will only see a reservation on a machine that can do the task, and will attempt to schedule the task for some other time when the machine is free. This will probably be something of a nuisance. One way of fixing this is to add a task field to the reservation record, and demand that the user fills this in when reserving a time slot. This can either be a permanent slot or a part of the advice field, but requires a change in the reservation making program.

The database access is extremely slow. It is necessary to go through the entire log history for every machine to ascertain the current machine states, and to traverse both the processflow and task trees of a lot to figure out the current location of a lot within its process. The simplest dispatch rules only need to know the remaining tasks on a lot, but scheduling rules for volume manufacturing (like the hedging point policy) need to be able to identify lots belonging to the same product type. Most of the time spent by the testbed is consumed when traversing task and processflow trees of the individual lots.

Certain data types, like hedging points, pertain to a particular opera-

[2]One can figure out the setup state in CAFE by going through the list of all completed operations in the lab to figure out which was the last operation performed on a machine, and from there infer the setup group and flow based on the input data. This is likely to be rather time consuming.

tion on every lot that belong to the same product type. For instance, this can refer to the third photolithography step on all DA lots, version 7. Such data are used to track and control overall production of this product type at particular stages in the process. CAFE has a distinction between tasks and processflows, where the task refers to an operation on a particular lot, whereas the processflow refers to the operation type in general and contains time required, machines that can do the operation, etc. Each task points to a processflow. However, CAFE uses shared processflows, so the photolithography steps along the process may be represented by the same data object. But the hedging points for the different steps must be different, since these are related to the remaining processing time on the lot. There does not seem to be a natural place to store data like hedging points and production surpluses in CAFE, but have set aside storage for these in the process description written to the resource file. When a process is taken from CAFE, these values will be initialized to zero. The actual values then need to be entered in the resource file by hand. The information entered will be lost whenever the resource file is rewritten from CAFE, for instance when a new process type has been introduced. This can be fixed permanently by either abolishing the shared processflows, such that each processflow object in CAFE describes one unique operation, or by introducing a third abstraction level between the task and the processflow for storing this kind of data only.

If a lot queues for an task that can be done on more than one machine, a lot arrival event will be generated at each of the machines. The first machine mentioned in the task's list of machines will receive this message first and will grab the lot if it is free. This may give an uneven loading of the machines. It might also be less than optimal if more than one machine is free, but there is a reservation on the first machine, causing the task to be delayed until after the reservation expires even if the other machine could load it immediately.

References

[1] S. Ahn, "Construction and implementation of real-time scheduling algorithms in a computer aided fabrication environment", Master's Thesis, MIT, February 1994.

[2] A. D. Baker and M. E. Merchant, "Automatic factories — How will they be controlled?", *IEEE Potentials*, pp. 15-20, December 1993.

[3] A. M. Bonvik, "Design proposal for the CAFE scheduling application", MIT CIDM Memo No. 94-6, 1994.

[4] G. Feigin, J. Fowler, J. Robinson, and R. Leachman, "Semiconductor Wafer Manufacturing Data Format Specification", Sematech, July 19, 1994.

[5] S. B. Gershwin, *Manufacturing Systems Engineering*, PTR Prentice-Hall, Englewood Cliffs, NJ, 1994.

[6] R. M. Harris and D. Boning, "Giraphe v3.3: A user's manual with examples", VLSI Memo No. 88-486, MIT, November 1988.

[7] P. R. Kumar, "Re-entrant Lines", Preprint, 1991.

[8] U. W. Pooch and J. A. Wall, *Discrete event simulation — A Practical Approach*, CRC Press, Boca Raton, FL, 1993.

[9] G. Sullivan and K. Fordyce, "IBM Burlington's Logistics Management System", *Interfaces*, **20**:1, pp. 43-64, January-February 1990.

[10] H. Yan, S. Lou, S. Sethi, A. Gardel and P. Deosthali, "Testing the robustness of various production control policies in semiconductor manufacturing", 1994, Forthcoming in *IEEE Transactions on Semiconductor Manufacturing*.

[11] G. van Ryzin, S. X. C. Lou and Stanley B. Gershwin, "Production control for a tandem two-machine system", *IIE Transactions*, **25**:5, pp. 5-20, September 1993.

A Sample resource file

```
HOURS
        Sun        00:00:00        00:00:00
        Mon        08:30:00        20:00:00
        Tue        08:30:00        20:00:00
        Wed        08:30:00        20:00:00
```

```
      Thu      08:30:00      20:00:00
      Fri      08:30:00      18:00:00
      Sat      00:00:00      00:00:00

HOLIDAYS
      09/05/94
      10/10/94
      10/11/94
      11/11/94
      11/24/94
      11/25/94
      12/25/94
      12/26/94

MACHINES
      ICL_tubeA5      37
            FAIL      gamma      0.527386      4637618.318242
            REPAIR    gamma      0.385637      1881224.427149
      ICL_ellipsometer      15
            FAIL      nil
            REPAIR    nil
      ICL_tubeA1      108
            FAIL      gamma      0.732355      7297763.812549
            REPAIR    gamma      0.369658      1651507.261803
      ICL_rca      90
            FAIL      gamma      0.419667      5913557.465355
            REPAIR    gamma      0.107754      2263459.933442

PROCESSES
        PROCESS      AMB-TEST_1      1214
            STARTS      exponential 5.787e-07
            SURPLUS     08/21/94 14:45:15      -100.0
            DEMAND      1.1574e-05
            PRIORITY    0
            WAFERCOUNT  20

            OPERATION      RCA-CLEAN      29269
                  OPSET      dsro430      27891
                  DURATION      7200
                  TRANSFERTIME      1800
                  HEDGINGPOINT      0
                  MACHINE      ICL_rca
            OPERATION      FURNACE-OPERATION      29270
                  OPSET      dsro430      27891
```

```
        DURATION      18000
        SETUP    FLOW    600
                 GROUP    group_a    1200
        TRANSFERTIME      1800
        HEDGINGPOINT      0
        MACHINE      ICL_tubeA1
  OPERATION      INSPECT-THICKNESS      29271
        OPSET      dsro430      27891
        DURATION      900
        SETUP    FLOW    300
        HEDGINGPOINT      0
        MACHINE      ICL_ellipsometer
  OPERATION      RCA-IONIC-CLEAN      29246
        OPSET      dnit1-5k      27895
        DURATION      7200
        HEDGINGPOINT      0
        MACHINE      ICL_rca
  OPERATION      DEPOSIT-MATERIAL      29247
        OPSET      dnit1-5k      27895
        DURATION      9000
        HEDGINGPOINT      0
        MACHINE      ICL_tubeA5
  OPERATION      INSPECT-THICKNESS      29248
        OPSET      dnit1-5k      27895
        DURATION      900
        HEDGINGPOINT      20
        MACHINE      ICL_ellipsometer
```

B Sample cache file

```
MACHINES
      ICL_rca               UP      07/27/93 14:02:37
      ICL_tubeA1            UP      07/27/93 14:02:00
      ICL_ellipsometer      UP      05/26/94 13:29:19
            RESERVATION      09/06/94 10:00:00 09/06/94 10:30:00
            RESERVATION      09/06/94 10:30:00 09/06/94 11:00:00
      ICL_tubeA5            UP      07/27/93 14:02:07

LOT      test_1_1      1850      ACTIVE
      PRIORITY 0
      WAFERCOUNT 20
      RELEASED 02/18/94 08:30:00
      TOUCHED  02/18/94 08:30:00
```

```
PROCESS AMB-TEST_1
TASK      RCA-CLEAN              27892      29269      READY
TASK      FURNACE-OPERATION      27893      29270
TASK      INSPECT-THICKNESS      27894      29271
TASK      RCA-IONIC-CLEAN        27896      29246
TASK      DEPOSIT-MATERIAL       27897      29247
TASK      INSPECT-THICKNESS      27898      29248
```

C Fab state display

```
---- Machine oriented state display 09/17/94 15:36:52 ----
Machine ICL_rca is UP since 07/27/93 14:02:37.
   No operation in progress.
   Lot test_1_1    op RCA-CLEAN    dur 2.00 h    since 02/18/94 08:30:00

Machine ICL_tubeA1 is UP since 07/27/93 14:02:00.
   No operation in progress.
   No waiting lots.

Machine ICL_ellipsometer is UP since 05/26/94 13:29:19.
   No operation in progress.
   No waiting lots.

Machine ICL_tubeA5 is UP since 07/27/93 14:02:07.
   No operation in progress.
   No waiting lots.


---- Lot oriented state display 09/17/94 15:36:52 ----
Lot test_1_1 is ACTIVE, size 20 wafers.
    Queued for operation RCA-CLEAN on ICL_rca
    Waited since 02/18/94 08:30:00.
    Surplus -127.04 at time 09/17/94 15:36:52, hedging point 20.00.
------------------------------------------------
```

D Performance measure examples

```
---- Machine oriented performance display 09/17/95 15:36:52 ----
Machine ICL_tubeA1
  Utilization:   1.48%      Setup time:    0.01%
  Down time:     2.18%      Idle time:    96.33%
```

```
  Avg queue:     0.062 lots

Machine ICL_rca
  Utilization:   1.19%      Setup time:    0.00%
  Down time:     0.15%      Idle time:    98.66%
  Avg queue:     1.012 lots

Machine ICL_tubeA5
  Utilization:   0.74%      Setup time:    0.00%
  Down time:     0.21%      Idle time:    99.05%
  Avg queue:     0.038 lots

Machine ICL_ellipsometer
  Utilization:   0.15%      Setup time:    0.02%
  Down time:     0.00%      Idle time:    99.83%
  Avg queue:     0.000 lots

---- Process oriented performance display 09/17/95 15:36:52 ----
Cycle times in weeks for process AMB-TEST_1
N 26  Avg 3.401  Std 5.709  CV 1.678  Min 0.266  Max 30.7
    Average queuing: 97.90%, processing: 2.10%
    Ratio (actual cycle time)/(raw processing time): 47.6185

------------------------------------------------
```