UNICOS under UNICOS: A Valuable Maintenance Tool !?

Mathilde Romberg, Zentralinstitut fuer Angewandte Mathematik (ZAM), Forschungszentrum, Juelich GmbH (KFA), Juelich, Germany

ABSTRACT: *KFA used UNICOS under UNICOS to upgrade to UNICOS 9.0. Most tests were conducted successfully in the guest system which reduced dedicated system time significantly. The first part of this paper covers KFA's very positive experience with UNICOS under UNICOS and gives hints and tips for system administration. A second part addresses general maintenance strategies for UNICOS on PVP and for UNICOS/mk on MPP systems.*

Introduction

To perform software maintenance on Cray systems it was common to reserve several hours of dedicated machine time each month or even each week. This time is required to ensure that the new (update) versions of software work properly. There was –and for some systems still is– no other technique to accomplish software maintenance. KFA normally used two to three hours per week to analyze software problems and test UNICOS update versions and twice that time while preparing a new UNICOS major release for production. In addition hardware maintenance required about two hours per week on average.
A strategy like this is very expensive since production has to be stopped even though the tests need only a fraction of a single CPU. The systems are getting bigger and bigger and as a result the problem of wasting money, i.e. expensive system time, becomes more serious. In addition, tracing (serious) problems can take very long because system administrators often have to wait one week or even longer for the next scheduled system time to install and check trace codes or updates. With 'UNICOS under UNICOS' (UuU) there exists a technique now to test software products and xes in parallel to production on some Cray systems. This is a feature known from other operating systems in the past like IBM's VM, which supported virtual machines and allowed to test different operating systems (VM, MVS, VSE) in parallel. This guest support for IBM operating systems was extended for better performance through microcode assist and eventually moved into hardware with IBM's PR/SM feature of ES/9000 models to allow multiple production systems to share one machine in a exible way with near-native performance.

KFA rst installed 'UNICOS under UNICOS' to prepare the upgrade to UNICOS 9.0 in summer 1995. The next section describes the feature and our experiences with this feature.

UNICOS under UNICOS

UNICOS under UNICOS implements the capability to run a UNICOS guest operating system under a native UNICOS system. The guest system is independent from the host except for I/O which is handled by the host IOS.

1 Requirements

The UuU feature is developed for Cray systems with a non-VME-based I/O subsystem, that means it can be used on machines which support IOS Model C, D or E, but not on CRAY EL or current J90 systems. It was introduced with UNICOS 7.0.7 and UNICOS 8.0.3. To run UuU the following resources are required: Disk space for separate *root*, *usr*, *src*, *tmp* and *swap* le systems, an additional network interface (or a NSC N130 adapter which can be shared), and at least 8 MW of main memory dedicated to the guest. The amount of main memory increases with the total memory size of the system, i.e. on a 256 MW machine the guest should be con gured with 16 MW since the kernel is normally built for the large (host) machine. The guest system is not capable to work with USCP and the shared le system (SFS).

UNICOS offers the ***guest*** command to control the guest system from the host: It is used to start-up and shutdown the guest and display information about it.

2 Con guration in General

After having done all necessary preparations[1] (i.e. additional le systems established on the host) one has to enable UuU in the host's installator and to specify the size of the guest's main memory, its CPU share, and the users that are allowed to start-up a guest system together with their guest environment. The guest start-up is thereby not tied to the *root* userid. For each of the speci ed users the host con gurator creates a directory */usr/guest/<userid>* which contains the guest system start-up le (*guest.rc*) for this user, the parameter le (*param*), the kernel (*unicos*), the *crash* and *kcompress* command, as well as the path of the *dump* directory. The start-up le has to

[1] "UNICOS under UNICOS Administrator's Guide", Cray Research, Inc. SG-2156

be customized to one's needs. It contains speci cations of memory size, network node name and the paths where *param*, *unicos*, *crash*, etc. are located. This allows to put these les into any directory (i.e. a well de ned place for all privileged users) and not necessarily into */usr/guest/<userid>*.

The parameter le should be a copy of the host's *param* le but without *memory* and *CPUS* speci cation. And in case the host uses a part of the main memory for *ldcache* the *LDCHCORE* speci cation for the guest has to be set explicitly to 0. These preparations are all done in the host system under which the guest will be run.

Figure 1 shows an example of the 'Guest Defaults' page of the con gurator. To enable the guest feature one guest system must be allowed. The panel suggests that a host can run multiple guests but up to now only one is supported. The 'maximum TTY connections' speci es the number of console lines dedicated to the guest (i.e. *zip* connections on a Model E IOS). As 'CPU allocation scheme' *MEMORY* or *DEFAULT* can be chosen. The option *DEFAULT* allows to specify percentages of the CPU for host and guest while *MEMORY* de nes a xed percentage of the CPU for the guest equal to the ratio of guest to host memory.

```
                         Guest Defaults

              Miscellaneous Options

    Maximum number of guest systems allowed.
      A value of zero (0) will DISABLE all
S->   guest(8) command functions.              1

      NOTE: To disable the guest feature at system boot time,
            set GUESTMAX to zero (0) in the UNICOS Kernel
            Configuration.

    Enable GUEST/HOST kernel tracing?          NO
    Halt HOST when guest panics?               NO
    Maximum TTY connections per guest          2
    Create user directories below /usr/guest?  YES
    Remove non-valid directories in /usr/guest? NO

              Memory Size Options

    Minimum HOST memory in megawords          400
    Minimum GUEST memory in megawords          32
    Maximum GUEST memory in megawords          32

              CPU Percentage Options

    CPU allocation scheme                      MEMORY
      Minimum HOST CPU percentage
      Maximum GUEST CPU percentage
    Minimum number of CPUs to remain in the host 0

    Reset global guest defaults...
```

Figure 1 Guest Defaults page of host con gurator

The following steps need to be done in the le systems of the guest: The con guration datasets like *fstab*, *daemons*, *rcoptions*, *gated.conf*, etc. and the local start-up scripts *rc.pre*, *rc.mid*, and *rc.pst*, *netstart.pst*, etc. have to be adapted. Most of these datasets can be kept in two versions, a host and a guest version marked by the suf x *.host* respectively *.guest*. They are linked to the of cial name (the le name without suf x) at start-up depending on the system (host or guest) started. This process is controlled by */etc/brc.guest*. Messages at host and guest start-up show this action (see gure 2). To

```
.
.
Checking /dev/dsk/root before running /etc/brc.guest.
/root: file system opened
.
.
/root: ***** FILE SYSTEM WAS MODIFIED *****

Executing /etc/brc.guest.
Linking Guest configuration files for HOST system startup:

   No filenames exist in /etc/config/guest_config

gencat: Starting pass 1
.
.
```

Figure 2 Messages at host system start-up

enable the system to exploit this feature the names of the versioned con guration datasets have to be speci ed in the */etc/con g/guest_con g* le. The advantage is that the same root le system can be used to start the native system as well as the guest (i.e. the "old" guest can become the new host during a system upgrade).

The guest system appears to be a separate machine with its own network addresses and le systems, even from the host system's point of view. It is impossible for the host to access a guest le system directly, for instance by mounting or dumping it, while the guest is running. This will result in warning messages on the console and the access will be denied (see gure 3). The same is true for the guest, therefore if le systems have to be accessed by both systems in parallel, NFS has to be used. All the resources (memory and CPU)

```
11:07:10 uts/c1/os/guest.c-02: WARNING  ovl: guest \
                                     configuration overlap

Open request from zam013
Conflicting resource belongs to m94guest
Device type:    Disk
        subtype DA302 (19)
        cluster        0
        iop            1
        channel        034
        unit           3
        starting block  1051344
```

Figure 3 CRAY console warning message "con icting resources"

de ned for the guest system are used by the host while the guest is not running. At guest start-up the de ned memory is dedicated to the guest, the host cannot use it until the guest is properly shutdown. While the guest system is running it gets the CPU cycles needed up to the de ned limit. All unused cycles can be used by the host.

3 Tested Configuration at KFA

KFA used the following con guration during the UNICOS 9.0 test on a CRAY M94/4512:
The host operating system level was 8.0.4 while the guest was at 9.0.1. The guest was given 32 MW of main memory. The le systems *groot*, *gusr*, *gsrc*, *gtmp*, and *gswap* occupied about 10 GBytes of disk space. All other le systems, especially those containing user data and local software, were NFS mounted from the host. An existing Ethernet backup connection was dedicated to the guest. As the CPU allocation scheme we initially used *MEMORY* but switched to *DEFAULT* later. *DEFAULT* is more exible since it allows to specify minimum host and maximum guest CPU percentages. Changes to this can be done while the host is active; neither a host nor a guest reboot is required.

4 Experiences

First of all we expected negative side effects for the host when running a guest (like crashes of the host as a result of problems in the guest) — but up to now there were none. The guest system crashed several times for different reasons but in no case the host was affected. Host and guest are completely isolated from each other.
The guest, of course, needs resources from the host, this can

```
          %usr   %sys  %wsem  #locks   %idle   %wio  %guest
12:15:01    94      6      0      63       0      0       0
12:35:00    94      6      0      55       0      0       0
12:55:01    92      7      1     222       0      0       0
13:15:00    87     13      3     676       0      0       1
13:35:00    91      9      1     325       0      0       0
13:55:00    94      5      0     179       0      0       1
14:15:01    93      6      1     239       0      0       1
14:35:00    93      6      1     216       0      0       0
14:55:00    89      9      1     246       0      0       2
```

Figure 4 Host's *sar* values, when the guest is running with low activity

```
          %usr   %sys  %wsem  #locks   %idle   %wio  %guest
14:00:02    84     10      2     730       0      0       7
14:15:01    83     10      1     394       0      0       6
14:35:01    84     12      1     434       0      0       4
14:55:01    81     12      1     384       0      0       7
15:15:02    82     11      2     531       0      0       7
15:35:00    85     11      1     467       3      0       1
15:55:00    89     11      1     423       0      0       1
16:00:01    89     11      1     383       0      0       0
```

Figure 5 Host's *sar* values, when the guest is running with CPU allocation scheme *MEMORY* and is highly active

be seen in the host's *sar* which shows increased system time, *%wsem*, and *#locks* while the guest is busy (see gures 4–7). *sar* also shows the part of the CPU time the guest consumes: With the allocation scheme *MEMORY* the guest gets about 7% of the system at most (see gure 5) which corresponds to the ratio of guest to host memory (32:480). With the allocation scheme *DEFAULT* it almost gets the de ned share (see gures 6 and 7). In both cases the minimum share for the host has been set to 100 – max. guest (70% resp. 85%).
The host is affected at system shutdown: If the guest is running, */etc/shutdown* in the host system sends a message to the system console saying the guest is running and waits for a reply whether to proceed or not. This is very disrupting in case of automatic operation (i.e. an automatic shutdown initiated by the OWS after detecting serious problems on the host) which we use intensively[2].
As we do not use tapes with the guest system, the backup of its le systems must be done in the host which requires the guest to be not running. Therefore a shutdown of the guest is needed to dump its le systems.

```
          %usr   %sys  %wsem  #locks   %idle   %wio  %guest
15:00:03    57     18      4    1095       0      0      25
15:15:01    68     12      2     495       0      0      20
15:35:00    64     12      2     486       0      0      24
15:55:00    59     17      3     874       0      0      24
16:15:02    59     23      7    1585       0      0      18
16:35:01    62     20      5    1249       0      0      18
16:55:00    67     13      3     645       0      0      21
17:00:01    60     16      3     783       0      0      24
```

Figure 6 Host's *sar* values, when the guest is running with CPU allocation scheme *DEFAULT* (max. guest percentage 30%) and is highly active

```
          %usr   %sys  %wsem  #locks   %idle   %wio  %guest
10:00:00    81      8      2     676       0      0      11
10:15:01    85      5      1     274       0      0      10
10:35:00    80     11      2     859       0      0      10
10:55:00    80      8      1     482       0      0      12
11:15:00    77     11      2     769       0      0      12
11:35:00    77     12      2     770       0      0      11
11:55:00    75     14      2     945       0      0      11
12:00:01    80      9      1     613       0      0      11
```

Figure 7 Host's *sar* values, when the guest is running with CPU allocation scheme *DEFAULT* (max. guest percentage 15%) and is highly active

During the UNICOS 9.0 test we did not check the functions of the tape- and the data migration daemon within the guest but

[2] "Automatic Supervision of CRAY UNICOS Systems" by N.Attig, V.Sander, L.Wollschlager and R.Krotz in Proceedings of the CRAY User Group Meeting (Spring), March 1991, London, pp.279–284.

in dedicated time. Every other needed component (TCP/IP, NQS, NFS, NIS, named, sendmail, ...) was tested in the guest except for USCP as it is not supported by the guest. We found some problems within the new operating system version, most of them corresponding to local con guration and changes in the system behavior. But we did not nd out a critical bug which only occurred with production load: After upgrading the host to UNICOS 9.0 we ran into system hang situations which could only be "solved" by a system crash (there were seven unscheduled interrupts due to this). The reason for this was an inadvertently (under heavy load) zeroed pointer to a kernel structure.

The users could participate more in testing than during the preparation phases of former major UNICOS releases. This time we could offer a separate test system for two weeks and not only for two or three dedicated maintenance periods of about four hours. Main users got the permission to log into the guest system and had the chance to test their production codes with UNICOS 9.0.

In comparison to the upgrade from UNICOS 7.0 to 8.0 when we had nine times the dedicated system for about four hours each we now needed only one dedicated period of four hours. In addition the guest used 168 CPU hours during the test phase, which is quite more than 9×4×4 CPU hours used before. This difference is due to a much larger user participation. In the past two xed test periods were allowed and this time the users had access to the test system for a period of two weeks. Looking back it was not necessary to give such a big portion (25%) of the system to the guest, 10% would have brought the same test result.

From a system administrator's point of view the biggest advantage during the UNICOS 9.0 test phase was the permanent access to a system running the new release. In case of problems we could analyze them carefully without wasting expensive dedicated time and con guration changes could be made and tested at once. This saved a lot of CPU cycles which could be used by the applications in the host, we needed about seven CPU hours instead of 6×4×4.

One of the nicest side effects of UuU is that a dump of the host system can be taken during production time without crashing the host using the *guest -D* command. It allows to dump all active systems; in case the guest is not up only the host is dumped. This helped us tracing serious problems in the host.

Requirements for a UNICOS Test Environment

When Cray systems were number crunching backends used exclusively in batch mode the hours dedicated to software maintenance were expensive but did not affect users seriously. Now that Cray systems have evolved over the last years to highly visible application servers, le servers, compute partner in interactive client/server software (UniChem for example), etc. the impact on the users becomes more critical.

The expectation for the system uptime of application servers is almost 24 hours a day, seven days a week all year long. Users often see this availability in other environments so Cray systems need to keep up.

As a matter of fact, state of the art in software maintenance has not kept pace. There are still many hours needed to test new versions or updates. Cray has made a major step in the right direction with the development of UNICOS under UNICOS as our experience has shown. It needs additional resources but it is a robust, exible, and an "exactly-what-is-needed" tool.

Looking at other systems and their maintenance strategies there are:

- workstation clusters, which allow tests on dedicated machines because they are cheap;
- parallel systems, which allow different versions of the operating system on different nodes;
- IBM's VM, which is comparable to UuU; and
- IBM's PR/SM and LPAR for hardware partitioning.

In some countries labor regulations may allow to perform software maintenance on Sunday night which can minimize the impact on the users, but most European countries and the unions forbid such a work schedule.

It is quite acceptable to have different software maintenance techniques in different environments but what is essential is that there **exists** one for **every** environment from the very beginning. Especially new software is likely to be less stable and requires more debugging and maintenance.

Therefore the following is needed:

- For all new PVP systems (i.e. T90, J90 with GigaRing) and their corresponding releases UNICOS under UNICOS must be supported;
- For the MPP system T3E an equivalent function must exist, e.g. partitioning with different operating system versions.

With the T90 Cray offers the logical machine feature which allows to split a hardware system in up to four logical machines. The feature requires dedicated I/O clusters and dedicated CPUs per logical machine and the memory can only be spilt into equal parts. The partitioning and con guration changes are done at system start-up. It is a very in exible feature and too expensive for software maintenance.

Conclusion

The UNICOS under UNICOS feature is an excellent solution for software maintenance. It is a highly needed, state of the art feature which meets our requirements for high system availability and low costs – but what is its future?

At this point in time there are no known plans whether UuU or an equivalent maintenance technique will be supported with the next generation of I/O subsystems, the GigaRing, for both PVP and MPP systems. But for software maintenance an equivalent tool is indispensable!