From individual scripts to the operational suite, its management, supervision and maintenance

Martin Janoušek
RC LACE, CHMI

Introduction

- Operational short-range NWP application has became an affordable technological flagship for every meteorological service.
- software + hardware + organization
- common features + specific particularities at every implementation

Objective of the lecture

- identify the common truths (even if they are obvious)
- offer the experience from ALADIN/LACE operations
- highlight a few interesting technical details
- invite for asking questions or to a visit

General requirements on a NWP model operations

- meteorologically correct and reliable
- technically stable – cron is not enough!
- products just in time
- state-of-art NWP application
- manageable, easy to maintain and extend

Steps to good design

- have plenty of time or forgiving users
- make a complete list of all applications covered by the suite
- make an agreement with the key users on the responsibility distribution
- determine delivery deadlines for key products
- choose technical solutions which are stable and under your control
- collect all items and test them
- design a suite, do a lot of experimentation
- do a re-engineering every 3-5 years

The suite elements

- model executable(s)
- namelists and climate files
- auxiliary files for canari/*n*DVAR
- file conversion tools (fa2grib, fa2temp, fa2netcdf, fa2rodos,…)
- post-processing tools (add diagnostic fields, EDF, blending family tools, oulan/bator/mandalay, chagal/ascs/…
- auxiliary tools (decdate, aft_time, frodo…)
- basic scripts

Model executable

- make one executable for all operational configurations
- compile first with a “safe” optimization, then aggressive optimization
- archive the full sources, libraries, makefiles, compilation logs
- well validated and stable

Modifications of the model code (1)

```
#!/usr/bin/ksh

NOMRESS=$1
echo $NOMRESS 'requested at ' 'date -u'
...
# look up the file
while [ ! -r $NOMRESS ]
do
  if [ $(hour70) -gt $STOPTIME ] ; then
       echo 'Too late for morgane'
       exit 130
   else
       sleep 30
   fi
done
...
```

Modifications of the model code (2)

- s

```
#!/usr/bin/ksh
...
NOMRESS=$*
echo $NOMRESS ’ announced at ’ ‘date -u‘
...
n=$(echo $NOMRESS | wc –c)
n1=$(( $n – 2 ))
n2=$(( $n – 1 ))
range=$(echo $NOMRESS | cut -c ${n1}-${n2})
...
if [ "`echo $NOMRESS | cut -c-5`" = "ICMSH" ] ;
then
  ${SMSBIN}/smsmeter hour ${range}
  $EVENT icmsh$range
fi
...
```

!*

CA

…

IN

IF(

EN

Model validation

- technical and scientific: should the norms change?
- integration 001: 10-30 days run + verif.pack + difficult cases
- e927: new 927 + old integration -> compare norms
- FULLPOS: on EDF bases

Auxiliary files

- namelists, climate files, auxiliary files for canari and nDVAR
- they should be on local disc
- test then explicitly with the operational executable

Scripts

- Korn shell
- invest to learning – advanced features
  - pipes
  - return-code-dependent branching (||, &&)
  - signal traps
  - use internal shell functions (arithmetics)
  - "subroutines“
- anything what can be changing with time or context should be a variable (date, namelists, executables,…)
- everything should be documented (inside the script – hashed text or in sms-manual style)

From a simple script to a suite task (1)

```
#! /usr/bin/ksh
# @$-lM 880mb

cd $TMPDIR
cp somewhere/ALADIN .
cp somewhere/morgane.namel fort.4
cp somewhere/ELSCFALADALBC??? .
ln -s ELSCFALADALBC000 ICMSHALADINIT
./ALADIN -ft${NSTOP} >lola
cat lola
cp ICMSHALAD+00?? somewhere
```

From a simple script to a suite task (2)

```
#! /usr/bin/ksh
# @$-lM 880mb
# @$-o %SMSNODE%:%SMSHOME%%SMSNAME%
# @$

%include <dateandtime.h>
%include <begin.h>

NSTOP=${NSTOP_morgane}
export STOPTIME=$(( ${STARTTIME} + ${TLIMIT3} ))

cd ${WORKDIR}/morgane

rm -f ifs* lola core* fort* ICMSHALAD+00*
cp ${BINDIR}/${ALADIN4morg} ALADIN
cp ${SXHOME}/bin/appel.moni.lace.morgane appel.moni.lace
cp ${SXHOME}/bin/signal.lace.morgane signal.lace

sed -e "s/_YYYYMMDD_/${YYYYMMDD}/" \
    -e "s/_NT_/$(( ${NT} * 3600 ))/" \
                $NAMDIR/${NAMEL.morgane} >fort.4

${EVENT} morganeINIT_b
${SMSBIN}/smslabel started "at ${DATUM}"

./ALADIN -ft${NSTOP} >lola
CODREP=$?

${SMSBIN}/smslabel finished "at" $( date -u)
${GZIPEXE} -c lola \
   >${LISTDIR}/morgane.${NEXP}.listing.${DD}${NT}.gz

( cd ../files
  for f in ../morgane/ICMSH${NEXP}+00??
        ../morgane/ICMSH${CNMEXP}INIT
  do
    ln $f $(basename $f).$YYYYMMDD$NT
  done ; )

for h in ${LANCRANGES} ; do  ${DESPATCH}
        ICMSH${NEXP}+00$h ; done

%include <end.h>
```

Input files download

- critical job - not fully controlled
- a great help if monitored by operators
- build a request file -> place it on pickup machine -> wait ±10s -> get the diagnostic file -> if OK, get the file, else wait 30s and retry
- ncftpget - control of frozen connections
- switch between pickup machines only if a put/get fails
- no complete consistency check - date check + ftp "Transfer complete." tracing

Remarks

- all the suite items must be stable and provide **reproducible results**
- save source code and makefiles
- "CVS everywhere"
- date-based naming + journal

Building a suite

Organization of supercomputer operations

- operations need absolute priority
- batch processing system (NQS, openPBS, LoadLeveler, LSM, …)
- hold/checkpoint or automated prioritizing?

An universal machine or specialized boxes?

- ALADIN/LACE:
  - a supercomputer (all ALADIN configurations, LACE products generating)
  - an universal unix server (suite control, file transfer, archiving, graphics)
- think about backup (install FULLPOS and product generation tools on some spare box)

Work out a suite structure: AOS

Setup : aps, sx4

Forec : getlbc
Blend : blend0
blend1
blend2
blend3
Lancelot : 00,03,…,48
morgane

Postproc: Fullpos : 00,03,…,48
Natsubgen: 00,06,…,48
Gribgen : 00,03,…,48
Rodos : 00,03,…,48
Chagal : 00,03,…,48

Serial or parallel suites?

- for fast lines and computers OR lack of time the suite can be **serial**:
  - download input files
  - pre-process them in parallel as much as possible
  - run the forecast with the full power
  - post-process in parallel
- **parallel** suites more efficient but complicated - need of reliable synchronization and filesystem organization

Synchronization of tasks

- by master shell and file tags
- by SMS
- combination

Tag synchronization suites

- first generation or RC LACE AOS
- a central shell process sets parameters and launches tasks (rather groups)
- synchronization of parallel tasks based on the time stamps and/or on tag files

+ simple, fast startup, easy to use for both tests and operational suites

- from some complexity hard to maintain, low granularity, cannot span multiple hosts

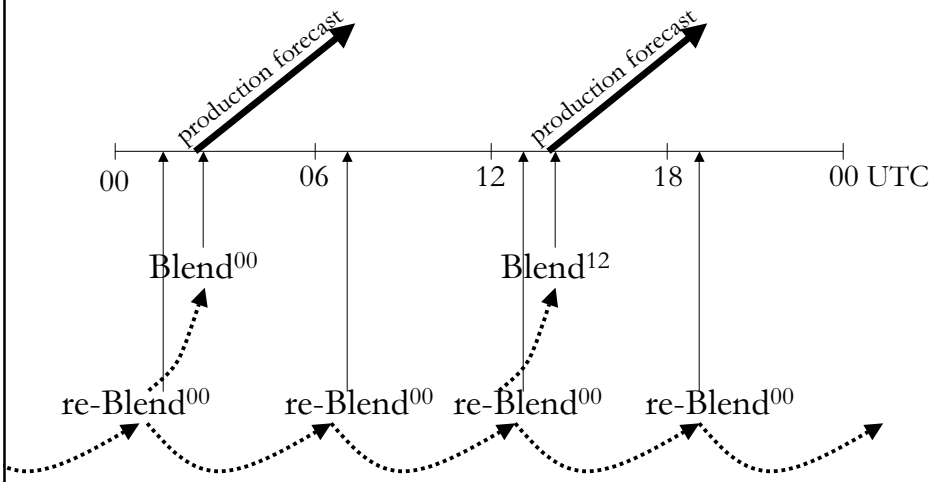
SMS-based suites

- Supervisor Monitor Scheduler, ©ECMWF
- scheduler of a large complex of computer tasks
- triggering of a task by time and/or by a status of other tasks
- tasks grouped to hierarchy of families and suites
- scheduler is controlled by user interface cdp and xcdp - monitoring and supervision

SMS-based suites (2)

- install SMS, learn the suite description language
- describe the suite, its tasks and their dependencies using the language
- place all scripts in a directory tree with the same structure like the suite families/tasks
- replace all % by %% and close comments between %manual, %endmanual pairs
- remove all shared items to include files (headers)
- use SMS-variables with moderation
- example: ALADIN/LACE

Suite cycling



Suite cycling (2)

- 2 families: Assim and Prod
- family Assim (blending)
  - at the end it increments the network time by 6h
  - it restarts immediately
  - call wait_till *network_time* + *late_cut-off_time*
- family Prod
  - starts at the early_cut-off_time
  - checks if the previous late-cut-off blending was finished
  - if it's running, wait for its completion
  - otherwise switch to dynamical adaptation

Transient files sharing

- all immediate products should be placed in a common repository ($WORKDIR/files) with the date and time suffix
- every task should link or copy a necessary file from the repository, eg.:
  - morgane: signal.lace copies ICMSHALAD+00xx to $WORKDIR/files and reports the file is ready
  - fullposXX job links the file ICMSHALAD+00xx as ICMSHALADINIT and runs FULLPOS

Dissemination of products

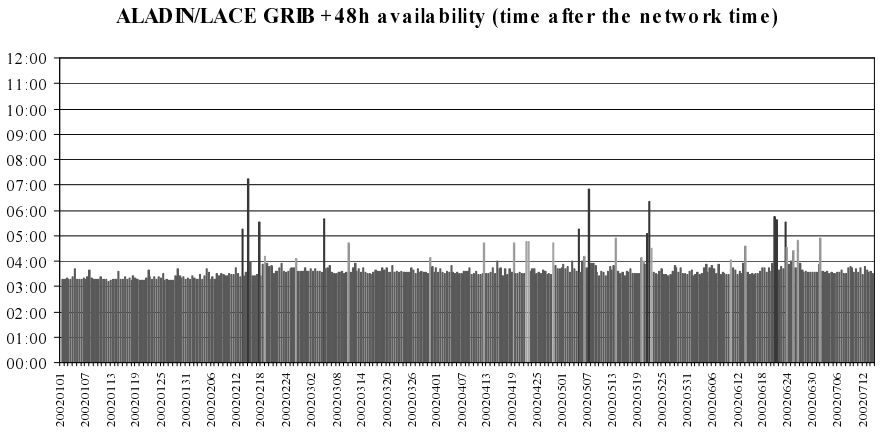
- dissemination queues + priority system
- separated from the rest of suite, a file distribution invoked by a single command
- NQS
- tqs (generated scripts + specialized daemon)
- afd (DWD)

Suite operations

ALADIN/LACE Operational Suite

- SMS-based suite + file tag control
- 24h/7d monitored by computer operators at checkpoints
- one AOS manager available on mobile phone
- no real-time automatic notification

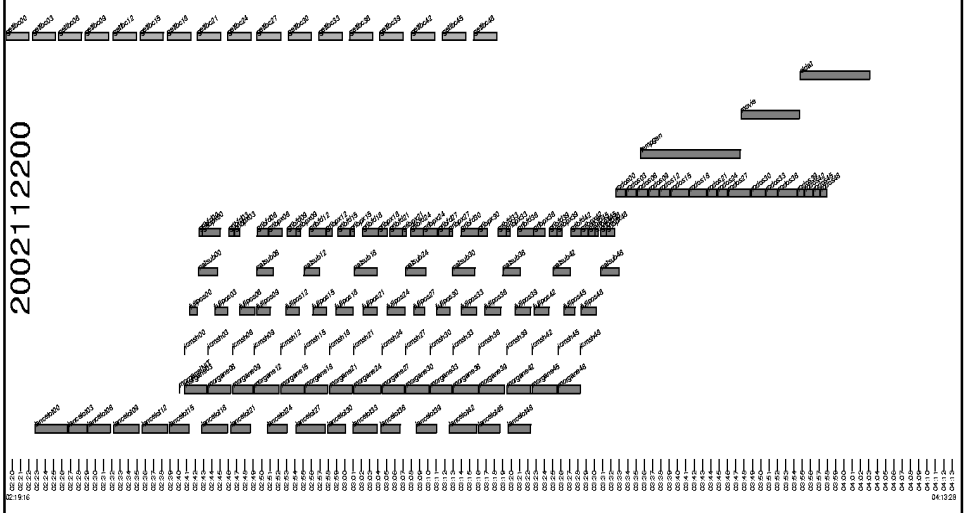
Production reliability



Suite changes

- nightmare in the living suites
- be suspecting - it's you who will be waken up
- golden "No-Friday-changes" rule
- for master script controlled suites a real testing should be possible

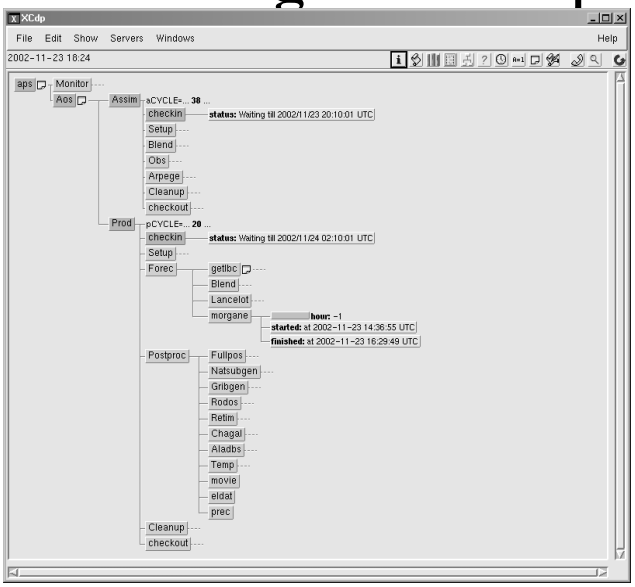
Suite balance



Monitoring tools

- time tags -> http://www.chmi.cz/meteo/ov/aladin/aos/aos state .
- WAP pages
- cdp2html
- aosc

Monitoring tools: Xcdp



Monitoring practices

- ALADIN is very stable - no need of a backup namelist
- make control plots - upper-air fields (also the troposphere), precipitation, wind field - chase instabilities
- watch forecast consistency - meteograms

Operator supervision

- no system is absolutely resilient to perturbations
- operator supervision is a big help, but the instructions must be unambiguous
- define clear responsibility
- identify checkpoints: times and what to check
  - check the line, incl. the distribution machine at M-F
  - check the status of local servers and NQS - hold the disturbing jobs
- do not rely on e-mail problem notification

Hotline support

- be a batch-system operator
- 90% of night calls are caused by our mistakes or lack of testing
- if the suite is stable there are 2 calls per month
- learn some French phrases

Last remarks

- Omitted topics
  - Parallel tests
  - Observation handling and data assimilation
  - Verification
- Cross backups