UL HPC Users' session: Mastering big data

Raymond Bisdorff

Université du Luxembourg
FSTC/ILAS

June 2018

Motivation: Showing a performance tableau

Consider a performance table showing the service quality of 12 commercial cloud providers measured by an external auditor on 14 incommensurable performance criteria.

criterion	upT	dwT	ouT	LB	MTBF	Rcv	Lat	RspT	Thrpt	stoC	snpC	auT	enC	auD
Amz	2	2	2	4	3	3	NA	3	NA	4	NA	4	4	4
Cen	4	4	0	4	4	4	NA	2	NA	3	NA	4	4	4
Cit	2	4	2	4	3	4	NA	2	NA	3	4	4	4	4
Dig	2	1	4	4	3	3	NA	2	NA	3	NA	4	4	4
Ela	4	4	0	4	4	4	NA	4	NA	3	4	4	4	4
GMO	1	3	4	4	3	2	NA	4	NA	3	NA	4	4	4
Ggl	4	2	1	4	2	3	NA	2	NA	4	4	4	4	4
HP	3	3	2	4	4	3	NA	4	NA	3	4	4	4	4
Lux	2	2	2	4	3	3	NA	2	NA	2	NA	4	4	4
MS	4	4	0	4	4	4	NA	4	NA	4	NA	4	4	4
Rsp	NA	NA	NA	4	NA	3	NA	NA	NA	3	4	4	4	4
Sig	4	4	0	4	4	4	NA	3	NA	3	4	4	4	4

Legend: 0 = '*very weak*', 1 = '*weak*', 2 = '*fair*', 3 = '*good*', 4 = '*very good*','*NA*' = missing data; '*green*' and '*red*' mark the **best**, respectively the **worst**, performances on each criterion.

Motivation: showing an ordered heat map

The same performance tableau may be optimistically colored with the highest 7-tiles class of the marginal performances and presented like a heat map,

Ranking of cloud providers by service quality

criteria	dwT	Rcv	MTBF	upT	RspT	stoC	auD	enC	auT	snpC	Thrpt	Lat	LB	ouT
weights	2.00	2.00	2.00	2.00	2.00	3.00	1.00	1.00	1.00	3.00	2.00	2.00	2.00	2.00
tau(*)	0.56	0.44	0.44	0.41	0.33	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.45
MS	4	4	4	4	4	4	4	4	4	NA	NA	NA	4	0
Ela	4	4	4	4	4	3	4	4	4	4	NA	NA	4	0
Sig	4	4	4	4	3	3	4	4	4	4	NA	NA	4	0
Cen	4	4	4	4	2	3	4	4	4	NA	NA	NA	4	0
HP	3	3	4	3	4	3	4	4	4	4	NA	NA	4	2
Cit	4	4	3	2	2	3	4	4	4	4	NA	NA	4	2
GMO	3	2	3	1	4	3	4	4	4	NA	NA	NA	4	4
Ggl	2	3	2	4	2	4	4	4	4	4	NA	NA	4	1
Rsp	NA	3	NA	NA	NA	3	4	4	4	4	NA	NA	4	NA
Amz	2	3	3	2	3	4	4	4	4	NA	NA	NA	4	2
Dig	1	3	3	2	2	3	4	4	4	NA	NA	NA	4	4
Lux	2	3	3	2	2	2	4	4	4	NA	NA	NA	4	2

Color legend:

quantile	20.00%	40.00%	60.00%	80.00%	100.00%

(*) tau: Ordinal (Kendall) correlation between marginal criterion and global ranking relation.

eventually linearly ordered, following for instance the Copeland ranking rule, from the best to the worst performing alternatives (ties are lexicographically resolved).

How to rank big performance tableaux ?

- The Copeland ranking rule, for instance, is based on crisp net flows requiring the in- and out-degree of each node in the outranking digraph;
- When the order n of the outranking digraph becomes big (several thousand or millions of alternatives), this requires handling a huge set of n^2 pairwise outranking situations;
- We use instead a sparse model of the outranking digraph, where we only keep a linearly ordered list of diagonal multicriteria quantiles equivalence classes with local outranking content.

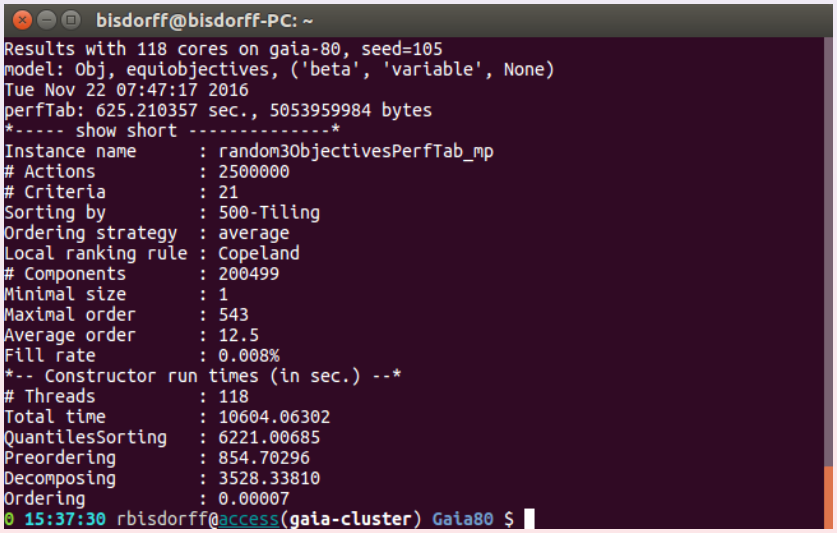
HPC performance measurements HPC school 2017

digraph order	standard model #c.	t_g sec.	τ_g	sparse model #c.	t_{bg}	τ_{bg}
1 000	118	6"	+0.88	8	1.6'	+0.83
2 000	118	15"	+0.88	8	3.5"	+0.83
2 500	118	27"	+0.88	8	4.4"	+0.83
10 000				118	7"	
15 000				118	12"	
25 000				118	21"	
50 000				118	48"	
100 000	(size	=	10^{10})	118	2'	(fill rate = 0.077%)
1 000 000	(size	=	10^{12})	118	36'	(fill rate = 0.028%)
1 732 051	(size	=	3×10^{12})	118	2h17'	(fill rate = 0.010%)
2 236 068	(size	=	5×10^{12})	118	3h15'	(fill rate = 0.010%)

Legend:

- #c. = number of cores;
- g: standard outranking digraph, bg: the sparse outranking digraph;
- t_g, resp. t_{bg}, are the corresponding constructor run times;
- τ_g, resp. τ_{bg} are the ordinal correlation of the Copeland ordering with the given outranking relation.

Gaia-80 November 2016 ranking record



New performance measurements Spring 2018

$\gtrapprox^q$ outranking relation order	size	q	fill rate	nbr. cores	run time
5 000	25×10^6	4	0.005%	28	0.5"
10 000	1×10^8	4	0.001%	28	1"
100 000	1×10^{10}	5	0.002%	28	10"
1 000 000	1×10^{12}	6	0.001%	64	2'
3 000 000	9×10^{12}	15	0.004%	64	13'
6 000 000	36×10^{12}	15	0.002%	64	41'

These run times are achieved both:

- on the Iris -skylake nodes with 28 cores,
- on the 3TB -bigmem Gaia-183 node with 64 cores, and
- running cythonized python modules in an Intel compiled virtual Python 3.6.5 environment [GCC Intel(R) 17.0.1 –enable-optimizations c++ 6.3 mode] on Debian 8 linux.

Successful actions for enhancing the performances - 1

- Algorithmic refinements: The pre-ranking quantiles sorting algorithm was further optimized, reducing considerably the fill rate of the sparse outranking digraphs;

Sparse versus standard digraph - Nov 2016

Symbol legend

- $\top$ outranking for certain
- $+$ more or less outranking
- ' ' indeterminate
- $-$ more or less outranked
- $\bot$ outranked for certain

Sparse digraph *bg*:
Actions : 50
Criteria : 7
Sorted by : 5-Tiling
Ranking rule : Copeland
Components : 7
Minimal order : 1
Maximal order : 15
Average order : 7.1
fill rate : 20.980%
correlation : $+0.7563$

Sparse outranking digraph - Now

```
*----- Object instance description -------------*
Instance class      : cQuantilesRankingDigraph
Instance name       : random3ObjectivesPerfTab_mp
# Actions           : 5000
# Criteria          : 21
Sorting by          : 5-Tiling
Ordering strategy   : average
Ranking rule        : NetFlows
# Components        : 4475
Minimal order       : 1
Maximal order       : 9
Average order       : 1.1
fill rate           : 0.008%
----  Constructor run times (in sec.) ----
Nbr of threads      : 8
Total time          : 1.63257
QuantilesSorting    : 1.23973
Preordering         : 0.02341
Decomposing         : 0.36922
Ordering            : 0.00000
```

Successful actions for enhancing the performances - 2

- **Algorithmic refinements**: The pre-ranking quantiles sorting algorithm was further optimized, reducing considerably the fill rate of the sparse outranking digraphs;
- **Reducing the size of python data objects**: A special bigData performance tableau model with integer dictionary keys and float evaluations is used for optimized Cython and C compiler variable typing;

Reducing the size of python data objects

tp1 Standard Random 3 Objectives performance tableau instance with 5000 decision actions and 21 performance criteria: $size(tp1) = 3\,602\,132$ Bytes.

tp2 Same BigData Random 3 Objectives performance tableau instance: $size(tp2) = 1\,398\,365$ Bytes.

Reducing the size of python data objects

tp1 Standard Random 3 Objectives performance tableau instance with 5000 decision actions and 21 performance criteria: $size(tp1) = 3\,602\,132$ Bytes.

tp2 Same BigData Random 3 Objectives performance tableau instance: $size(tp2) = 1\,398\,365$ Bytes.

bg1 Standard pre-ranked outranking digraph instance generated from tp1: $size(bg1) = 9\,471\,896$ Bytes.

bg2 BigData pre-ranked outranking digraph instance generated from tp2: $size(bg2) = 1\,791\,755$ Bytes.

Efficient Cython inline function declaration with variable typing

```
cdef inline int _localConcordance(float d, float ind, float wp, float p):
    """ None = -1.0 """
    if p > -1.0:
        if   d <= -p:
            return -1
        elif ind > -1.0:
            if d >= -ind:
                return 1
            else:
                return 0
        elif wp > -1.0:
            if d > -wp:
                return 1
            else:
                return 0
        else:
            if d < 0.0:
                return -1
            else:
                return 1
    else:
...
```


Successful actions for enhancing the performances - 3

- **Algorithmic refinements**: The pre-ranking quantiles sorting algorithm was further optimized, reducing considerably the fill rate of the sparse outranking digraphs;
- **Reducing the size of python data objects**: A special bigData performance tableau model with integer dictionary keys and float evaluations is used for optimized Cython and C compiler variable typing;
- **Efficient sharing of static data**: Global python variables allow to efficiently communicate static data objects to parallel threads when using -bigmem nodes;

Successful actions for enhancing the performances - 4

- **Algorithmic refinements**: The pre-ranking quantiles sorting algorithm was further optimized, reducing considerably the fill rate of the sparse outranking digraphs;
- **Reducing the size of python data objects**: A special bigData performance tableau model with integer dictionary keys and float evaluations is used for optimized Cython and C compiler variable typing;
- **Efficient sharing of static data**: Global python variables allow to efficiently communicate static object data to parallel threads when using -bigmem nodes;
- **Using a multiprocessing tasks queue**: Sorting tasks in decreasing durations and using an automatic multithreading mechanism (see the *multiprocessing python3 documentation*)

Using a multiprocessing tasks queue

```
with TemporaryDirectory(dir=tempDir) as tempDirName:
    ## tasks queue and workers launching
    NUMBER_OF_WORKERS = nbrOfCPUs
    tasksIndex = [(i,len(decomposition[i][1])) for i in range(nc)]
    tasksIndex.sort(key=lambda pos: pos[1],reverse=True)
    TASKS = [(Comments,(pos[0],nc,tempDirName)) for pos in tasksIndex]
    task_queue = Queue()
    for task in TASKS:
        task_queue.put(task)
    for i in range(NUMBER_OF_WORKERS):
        Process(target=_worker,args=(task_queue,)).start()
    if Comments:
        print('started')
    for i in range(NUMBER_OF_WORKERS):
        task_queue.put('STOP')

    while active_children() != []:
        pass
    if Comments:
        print('Exit %d threads' % NUMBER_OF_WORKERS)
```


Successful actions for enhancing the performances - 5

- **Algorithmic refinements**: The pre-ranking quantiles sorting algorithm was further optimized, reducing considerably the fill rate of the sparse outranking digraphs;
- **Reducing the size of python data objects**: A special bigData performance tableau model with integer dictionary keys and float evaluations is used for optimized Cython and C compiler variable typing;
- **Efficient sharing of static data**: Global python variables allow to efficiently communicate static object data to parallel threads when using -bigmem nodes;
- **Using a multiprocessing task queue**: Sorting tasks in decreasing durations and using an automatic multithreading mechanism.
- **Efficient UL HPC cluster equipments and staff**:
  Thank you for your support :)

Further documentation resources

Our cythonized Python HPC modules are freely available under the cython directory on:

- https://github.com/rbisdorff/Digraph3 and on
- https://sourceforge.net/projects/digraph3/

Tutorials and technical documentation + source code listings may be consulted on:

- https://digraph3.readthedocs.io/en/latest/