# Introduction to Using the BGQ

BGQ Day

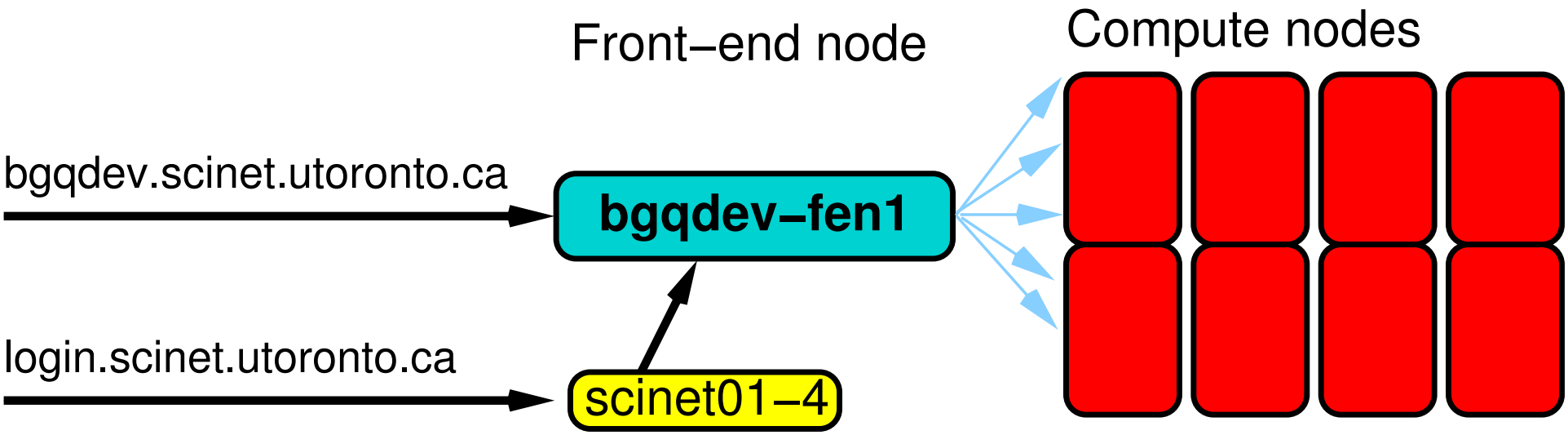
Toronto, Ontario

Jan 30, 2013/Mar 30, 2015

# Outline

1. Logging in
2. Data management
3. Software/Libraries
4. Compilers
5. Job scheduler
6. Performance considerations

# Get on the system




## Login must be through a secure shell (ssh)

```
ssh <user>@bgqdev.scinet.utoronto.ca
```

Can go through SciNet login nodes as well (`login.scinet.utoronto.ca`).

# Storage

## Your files on the BGQ systems

- BGQ front-end and compute nodes share one parallel file system.
- Compute nodes do not contain hard drives, but access this file system as well through a limited number of IO nodes.
- Each user has two directories on this file system:
  1. `/home/<first-letter-of-group>/<group>/<user>`
  2. `/scratch/<first-letter-of-group>/<group>/<user>`

  Example: `/home/s/scinet/rzon`, `/scratch/s/scinet/rzon`
  `<group>` is typically the PI's `<user>`.
- Environment variables `$HOME` and `$SCRATCH` contain these names.

# Storage - transfer

## How to get your files to BGQ?

- Must copy your code/data using 'secure copy'
- Simple examples:

```
scp <files> <user>@bgqdev.scinet.utoronto.ca:
scp -r <folder> <user>@bgqdev.scinet.utoronto.ca:<target>
```

- If the connection is slow, try adding the -C flag to use compression.
- To copy data from or to the SciNet file system, some SciNet nodes (p701, gpc01, gpc02, gpc03, gpc04) have the bgq home and scratch file systems mounted under /bgq/home and /bqq/scratch.

# Storage - limits

| location | purpose | quota | time-limit | backup |
|---|---|---|---|---|
| $HOME | development | 50GB | none | yes |
| $SCRATCH | computations | 20TB/1M | none currently | no |

## How much disk space do I have left?

The `diskUsage` command, available on the bgqdev nodes, provides information in a number of ways on the home and scratch file systems.

# On the front-end nodes

## You can

- See and manage your files
- Compile your application/library with special bgq compilers
- Write job submission scripts
- Submit your jobs to the scheduler
- Check job status

## But

- You cannot compile with gcc, g++, gfortran, nor with xlc, xlf, ...
- You cannot link with libraries in standard folders (`/lib64`, `/usr`)
- You cannot run your BGQ code
- You cannot see you SciNet files
- You cannot ssh to compute nodes (not even while running jobs)

# Software and Libraries

Once you log into devel nodes, what software is already installed?

- Other than essentials, all software installed as modules.
- modules set environment variables `LD_LIBRARY_PATH, PATH, ...`)
- Allows multiple, conflicting versions of package to be available.
- More on the BGQ page of wiki.

```
bgqdev-fen1-$ module avail
---------- /usr/share/Modules/modulefiles------
-
dot                 modules           use.own
module-cvs          use.deprecated    null
module-info         use.experimental

---------- /scinet/bgq/Modules/modulefiles
-----
BGW-paratec/1.0.4-2.0.0436
FEN/ncview/2.1.2
FEN/udunits/2.1.24
bgqgcc/4.4.6
binutils/2.21.1
cmake/2.8.8
compression/all
cxxlibraries/boost/1.47.0
ddd/3.3.12
essl/5.1
extras/bgqlinux
fftw/2.1.5
fftw/3.1.2-esslwrapper
fftw/3.3.2(default)
gdb/7.2
gnuplot/4.2.6
gnuplot/4.6.1(default)
...
```

# Software and Libraries

| | |
|---|---|
| `module load <module-name>` | use particular software |
| `module purge` | remove currently loaded modules |
| `module avail` | list available software packages |
| `module list` | list loaded modules |
| `module help <module-name>` | describe a module |

- Load frequently used modules in `.bashrc` in home directory.
- Short name gives default (e.g. `mpich2` $\longrightarrow$ `mpich2/xl`)
- To compile code that uses that a library from a module, add

  `-I${SCINET_[shortmodulename]_INC}`

- To link, add

  `-L${SCINET_[shortmodulename]_LIB}`

  (in addition to any `-l<libname>` arguments).

# Dependencies

- Modules sometimes require other modules to be loaded first.
- Module will let you know if you didn't.
- For example:

```
bgqdev-fen1-$ module purge
bgqdev-fen1-$ module load lapack
lapack/3.4.2(11):ERROR:151:  Module 'lapack/3.4.2' depends on one of
the module(s) 'mpich2/xl.ndebug mpich2/xl.legacy.ndebug mpich2/xl.legacy
mpich2/xl mpich2/gcc'
lapack/3.4.2(11):ERROR:102:  Tcl command execution failed:  prereq mpich2
bgqdev-fen1-$ module load mpich2
bgqdev-fen1-$ module load lapack
bgqdev-fen1-$ module list
Currently Loaded Modulefiles:
      1) mpich/xl      2) lapack/3.4.2
bgqdev-fen1-$
```

# Compiling with the IBM compilers

- Load the compiler module `vacpp` (for c/c++) or `xlf` (for fortran)
- Use `bgxlc,bgxlC,bgxlf90` for C, C++, and Fortran
- For OpenMP, use the thread-safe variants `bgxlc_r,bgxlC_r,bgxlf90_r`.
- For MPI, load an `mpich2` module first.
- Then use the mpi wrappers: `mpicc,mpicxx,mpixlf90` or `mpicc_r,mpicxx_r,mpixlf90_r`.
- Suggested compiler flags:

  ```
  -O3 -qarch=qp -qtune=qp
  ```

  supplemented by `-qsmp=omp` for OpenMP programs.

# Submitting jobs

## BGQ=shared resource

- To run a job, you must submit to LoadLeveler, a batch scheduler.
- You submit jobs from a front-end node in the form of a script
- In the script, use `runjob`, not `mpirun`.
- Scheduling is by 64-node block (i.e. 1024 cores)!
- Best to run from the scratch directory.
- Copy essential results out after your runs have finished.

# Queuing

Submitting is done with

```
llsubmit <script>
```

- The maximum walltime of any job is 24 hours.
  Longer runs should checkpoint.
- The minimum number of cores per job is 1024.
- The processors have 4 way SMT: **4**× as many 'virtual' cores.
- To make your jobs start sooner, reduce the `wall_clock_limit` to be closer to the estimated run time (perhaps adding about 10 % to be sure). Shorter jobs are scheduled sooner than longer ones.

# Job scripts

- A job script contains resource requests for the scheduler.
- Furthermore contains a script that calls runjob, e.g.

```
#!/bin/sh
# @ job_name = bgsample
# @ job_type = bluegene
# @ comment = "BGQ Job By Size"
# @ error = $(job_name).$(Host).$(jobid).err
# @ output = $(job_name).$(Host).$(jobid).out
# @ bg_size = 64
# @ wall_clock_limit = 30:00
# @ bg_connectivity = Torus
# @ queue

# Launch all BGQ jobs using runjob
runjob --np 1024 --ranks-per-node=16 --envs OMP_NUM_THREADS=1 \
--cwd=$SCRATCH/ :  $HOME/mycode.exe myflags
```

- Job scripts run on the front-end node. Only programs started with runjob run on the bgq. Keep scripting to a bare minimum.

# runjob

- All BGQ jobs are launced using runjob (analogous to mpirun).
- Jobs run on a block, which is a predefined group of nodes that have already been configured and booted by loadleveler.
- For example, if your loadleveler script requests 64 nodes, each with 16 cores (for a total of 1024 cores), you run a job with 16 processes per node and 1024 total processes with

```
runjob --np 1024 --ranks-per-node=16 --cwd=$PWD : $PWD/code file.in
```

- For pure mpi jobs, it is advisable always to give the number of ranks per node, because the default value of 1 may leave 15 cores on the node idle. The argument to ranks-per-node may be 1, 2, 4, 8, 16, 32, or 64.

# Job monitoring

Once your job is in the queue, you can monitor the queue:

- to see running jobs

```
llq2
```

(`llq -b` will work as well, giving slightly different information)

- to cancel a job use

```
llcancel JOBID
```

- and to look at details of the bluegene resources use

```
llbgstatus
```

# Interactive Use

- Beneficial when debugging and developing.
- A script has been written to allow a session in which runjob can be run interactively.
- Uses loadleveler to setup a block and set all the correct environment variables and then launch a shell on the front-end node.
- The debugjob session allows a 30 minute session on 64 nodes.

```
[bgqdev-fen1]$ debugjob
[bgqdev-fen1]$ runjob --np 64 --ranks-per-node=16 :  my_code myflags
[bgqdev-fen1]$ exit
```

- Also useful for building libraries and applications that need to run small tests as part of their 'configure' step. Can automatically run on the bgq (instead of the fen) by setting

```
[bgqdev-fen1]$ export BG_PGM_LAUNCHER=yes
[bgqdev-fen1]$ export RUNJOB_NP=1
```

# Sub-block jobs

- BGQ allows multiple applications to share the same block.
- To run a sub-block job, you need to specify a "`--corner`" within the block to start each job and a 5D Torus AxBxCxDxE "`--shape`".
- We have a script called subblocks that computes the `--corner` and `--shape` options for you given a block size.

```
source subblocks 4
for (( i=0; i< 16 ; i++)); do
  runjob --corner ${CORNER[$i]} --shape $SHAPE --np 64 \
  --ranks-per-node=16 :  your_code_here > $i.out&
done
wait
```

# Performance: File system

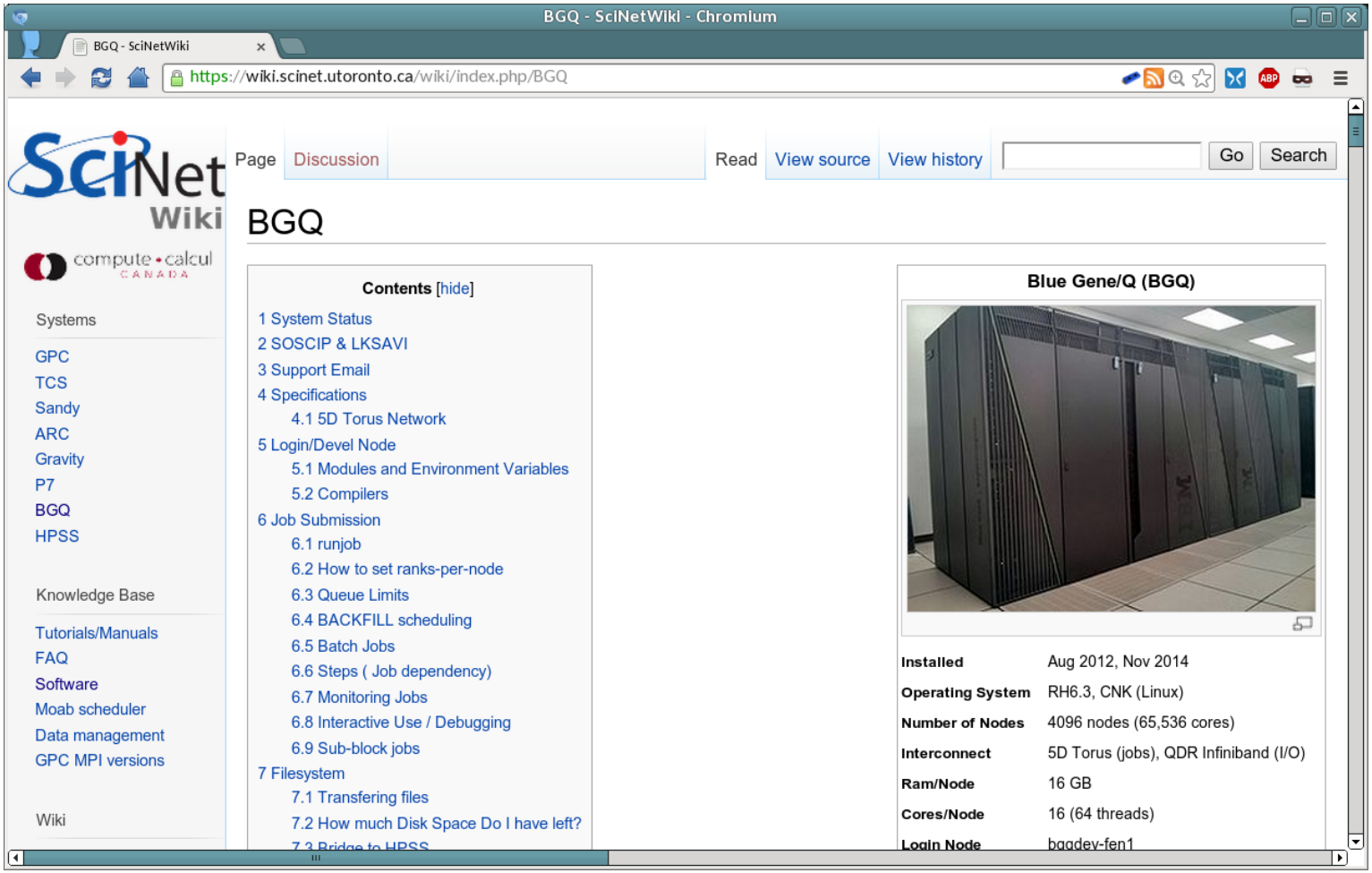
- Compute nodes do not contain hard drives!
- Any IO is going through a limited number of IO nodes (1 per 1024 cores).
- IO Nodes are mostly hidden from user code, but be aware.
- The available disk space, /home and /scratch, all part of the GPFS file system.
- GPFS is a high-performance file system which provides rapid reads and writes to large data sets in parallel from many nodes.
- It performs quite poorly at accessing data sets which consist of many, small files.

# I/O strategies

- Don't keep many small files on the system.
  They waste space, and are slower to access, read and write.
- Do not read and write lots of small amounts of data to disk.
  Reading data in from one 4MB file can be enormously faster than from 100 40KB files.
- Write your data out in binary. Faster and takes less space.
- Each process writing to a file of its own is not scalable.
  A directory gets locked by the first process accessing it, so the other processes have to wait for it.
- Consider using MPI-IO (part of the MPI-2 standard), (parallel) NetCDF or HDF5.

# More information and support

https://support.scinet.utoronto.ca/wiki/index.php/BGQ




Technical support: bgq-support@scinet.utoronto.ca