

GUROBI OPTIMIZER REMOTE SERVICES MANUAL



Version 8.0, Copyright © 2018, Gurobi Optimization, LLC

1	Introduction	4
2	Gurobi Compute Server and Remote Services Overview	5
2.1	Compute Server	6
2.2	Distributed Algorithms	9
2.3	Roles	11
	Cluster Administrator	11
	Administrator	11
	Client	11
2.4	Simple Example	12
3	Cluster Setup and Administration	13
3.1	Setting up Remote Services	14
	Installation	14
	Licensing	15
	Remote Services Agent (grb_rs)	15
	Configuration	16
	Starting Remote Services as a Process	18
	Starting Remote Services as a Service	19
	Verification	20
3.2	Forming a Cluster	22
	Connecting Nodes	22
	Compute Servers and Distributed Workers	24
	Grouping	25
	Processing State and Scaling	25
3.3	Communication Options	27
	Using HTTPS	27
	Using HTTPS with Self-Signed Certificates	27
	Firewalls	28
	Router	28
3.4	Maintaining a Cluster	30
	Managing Runtimes	30
	Upgrading Remote Services	31
4	Using Remote Services	32
4.1	Client Configuration	33
	License File	33
	Queueing, Load Balancing, and Job Priorities	34
	Submitting Jobs with gurobi_cl	34

	Issuing Cluster Management Commands: Using grbcluster	34
4.2	Client Commands	35
	Listing Optimization Jobs	35
	Accessing Job Logs	36
	Accessing Job Parameters	37
	Listing Cluster Nodes	38
	Troubleshooting Connectivity Issues	38
4.3	Administrative Commands	39
	Listing Cluster Licenses	39
	Changing the Job Limit	39
	Aborting Jobs	40
5	Programming with Remote Services	41
5.1	Using an API to Create a Compute Server Job	42
5.2	Performance Considerations on a Wide-Area Network (WAN)	43
5.3	Callbacks	44
5.4	Developing for Compute Server	45
5.5	Distributed Algorithms	46
5.6	Distributed Algorithm Considerations	48
5.7	Cluster REST API	49
6	Using Remote Services with Gurobi Instant Cloud	50
6.1	Client Setup	51
6.2	Client Commands	52
6.3	Administrative Commands	53
6.4	Region Router	54
7	Migrating from Previous Releases	55
7.1	Referring to Compute Servers	56
7.2	Installation - Ports and Firewalls	57
7.3	Installation - Encryption	58
7.4	Command-line options in gurobi_cl	59
7.5	Distributed Optimization	60
A	Appendix A: grb_rs	61
B	Appendix B: grb_rs - Configuration Properties	63
C	Appendix C: grbcluster	66
D	Appendix D: gurobi_cl	67
E	Appendix E: Acknowledgement of 3rd Party Icons	69
F	Appendix F: Open Source Component Licenses	70

Gurobi Remote Services is a set of Gurobi features that allow a cluster of one or more machines to perform Gurobi computations on behalf of other machines. The most powerful Remote Service is [Compute Server](#), which allows you to offload all Gurobi computations from a client machine onto a Remote Services cluster. A second Remote Service allows you to execute [distributed algorithms](#), wherein multiple machines can be used to accelerate a single optimization computation (e.g., solving a single MIP model, or performing automatic parameter tuning on a model).

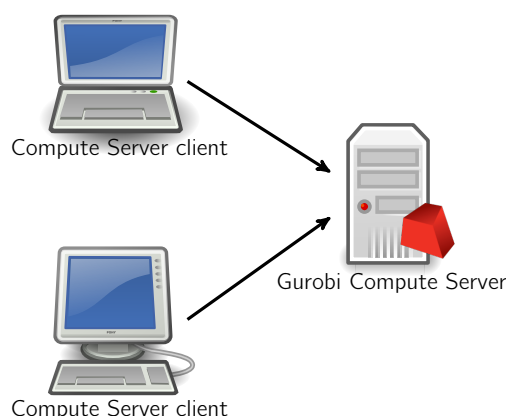
This document is organized into a number of sections. The first section provides an [overview of Gurobi Compute Server and Remote Services](#). The next section provides details on [setting up Remote Services](#). Then, the following sections provide details on [using Remote Services](#) and [programming with Remote Services](#). Finally, we discuss [using Remote Services with Gurobi Instant Cloud](#).

Gurobi Compute Server and Remote Services Overview

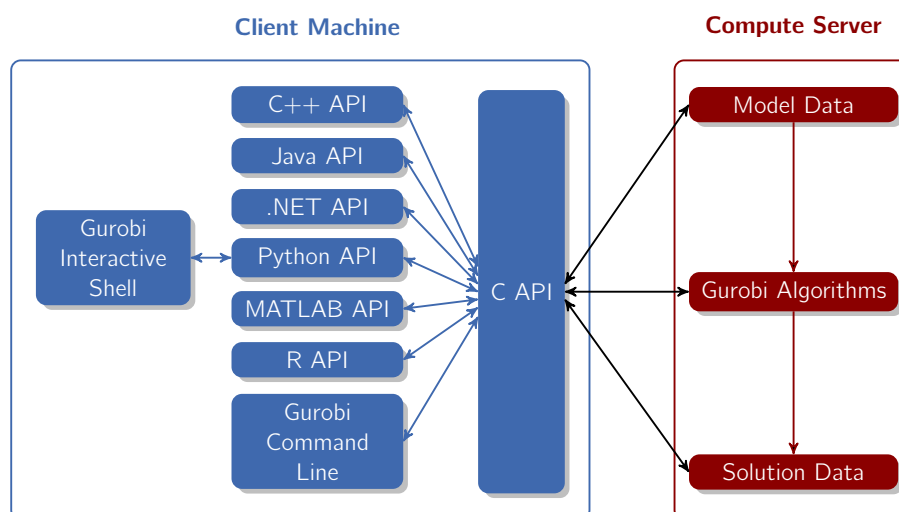
This section gives a quick introduction to the various capabilities of Gurobi Remote Services. It starts with a discussion of common use cases, and continues with a description of some of the more important Remote Services features. It then continues with a discussion of the different [roles](#) that typically come into play in a Gurobi Remote Services cluster. The client role is fairly simple to understand, but the administrative and cluster administrative roles require a bit more elaboration. We'll also talk about the [Remote Services agent](#), which is the program that runs on each Remote Services node, and [grbcluster](#), which is the program that is used to perform various administrative tasks. Finally, we give a [simple example](#) of how to submit a job from a client to a Compute Server cluster.

2.1 Compute Server

As noted earlier, Gurobi Compute Server is an optional component of [Gurobi Remote Services](#) that allows you to choose one or more servers to run your Gurobi computations. You can then offload the work associated with solving optimization problems onto these servers from as many client machines as you like:



When considering a program that uses Gurobi Compute Server, you can think of the optimization as being split into two parts: the client and the compute server. A client program builds an optimization model using any of the standard Gurobi interfaces (C, C++, Java, .NET, Python, MATLAB, R). This happens in the left box of this figure:

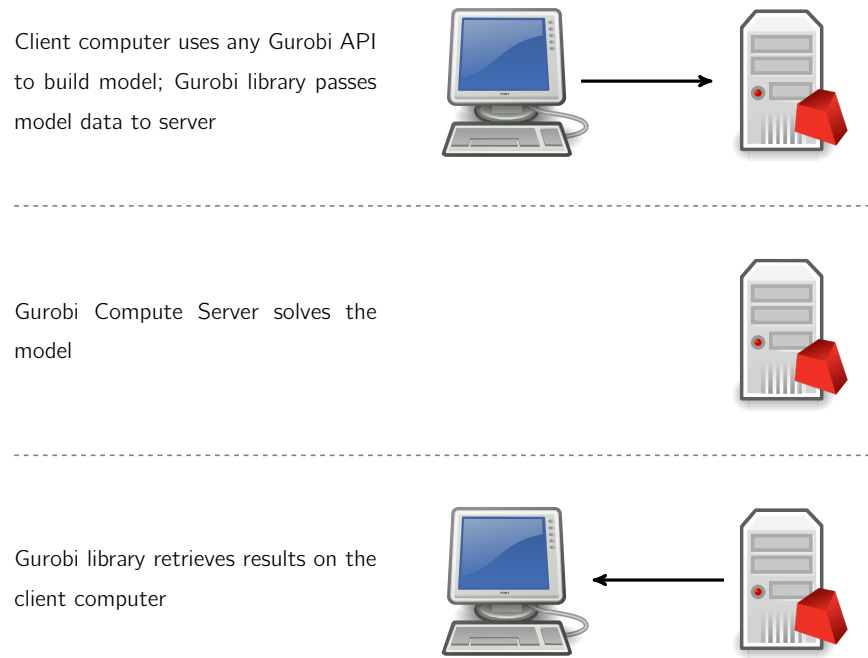


All of our API's sit on top of our C API. The C API is in charge of building the internal model data structures, invoking the Gurobi algorithms, retrieving solution information, etc. When running Gurobi on a single machine, the C API would build the necessary data structures in memory. The Gurobi algorithms would take the data stored in these data structures as input, and produce solution data as output.

When using a Compute Server, the C API instead passes model data to the server, where it is stored. When the Gurobi algorithms are invoked, the C API simply passes a message to the

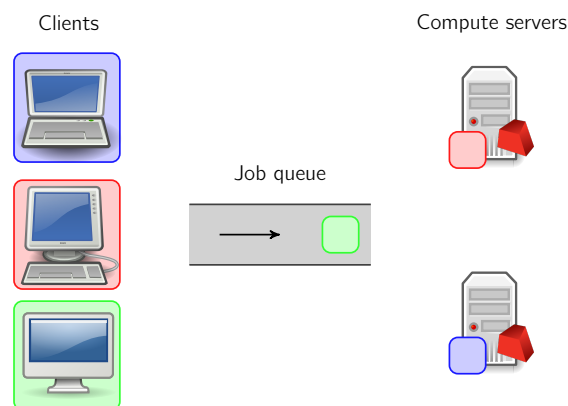
server, indicating that optimization should be performed on the stored model data. Solution data is computed and stored on the server. When the client program later queries the solution information, the client sends a message to the server in order to obtain the requested data. All communication between the client and server happens behind the scenes,

In other words, the overall process can be viewed as happening in three phases:



Of course, programs that use the Gurobi API's in more complex ways would have additional steps.

Gurobi Compute Servers support queuing and load balancing. You can set a limit on the number of simultaneous jobs each Compute Server will run. When this limit has been reached, subsequent jobs will be queued. If you have multiple Compute Server nodes configured in a cluster, the current job load is automatically balanced among the available servers.



By default, the Gurobi job queue is serviced in a First-In, First-Out (FIFO) fashion. However, jobs can be given different priorities (through a client license file, or through API calls). Jobs with higher priorities are then selected from the queue before jobs with lower priorities.

While the Gurobi Compute Server is meant to be transparent to both developers and users, there are a few aspects of Compute Server usage that you do need to be aware of. These include performance considerations, APIs for configuring client programs, and a few features that are not supported for Compute Server applications. These topics will be discussed [later in this document](#).

2.2 Distributed Algorithms

As noted earlier, Gurobi Optimizer implements a number of distributed algorithms that allow you to use multiple machines to solve a problem faster. Available distributed algorithms are:

- **A distributed MIP solver**, which allows you to divide the work of solving a single MIP model among multiple machines. A manager machine passes problem data to a set of worker machines in order to coordinate the overall solution process.
- **A distributed concurrent solver**, which allows you to use multiple machines to solve an LP or MIP model. Unlike the distributed MIP solver, the concurrent solver doesn't divide the work among machines. Instead, each machine uses a different strategy to solve the whole problem, with the hope that one strategy will be particularly effective and will finish much earlier than the others. For some problems, this concurrent approach can be more effective than attempting to divide up the work.
- **Distributed parameter tuning**, which automatically searches for parameter settings that improve performance on your optimization model (or set of models). Tuning solves your model(s) with a variety of parameter settings, measuring the performance obtained by each set, and then uses the results to identify the settings that produce the best overall performance. The distributed version of tuning performs these trials on multiple machines, which makes the overall tuning process run much faster.

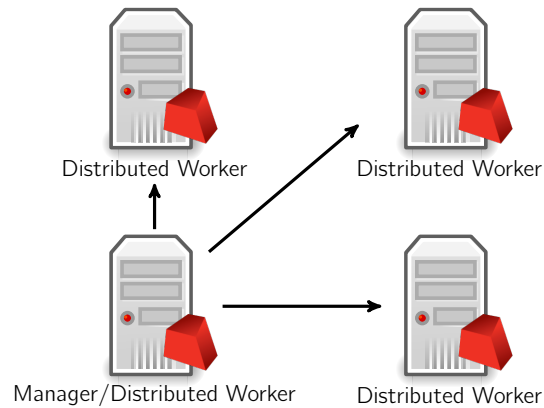
These distributed algorithms are designed to be nearly transparent to the user. The user simply modifies a few parameters, and the work of distributing the computation among multiple machines is handled behind the scenes by the Gurobi library.

Distributed Workers and the Distributed Manager

Running distributed algorithms requires several machines. One acts as the manager, coordinating the activities of the set of machines. The others act as workers, receiving tasks from the manager. The manager typically acts as a worker itself, although not always. More machines generally produce better performance, although the marginal benefit of an additional machine typically falls off as you add more.

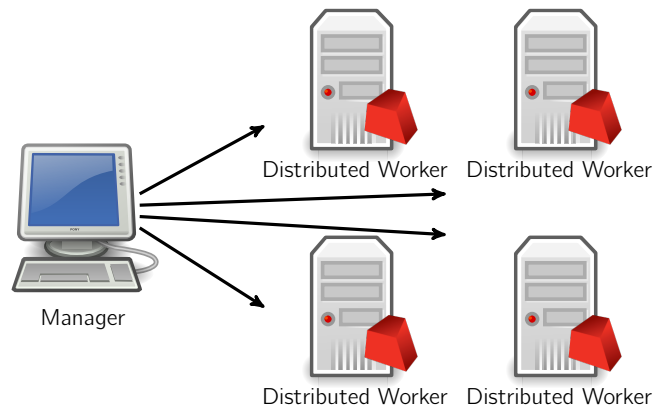
As we'll discuss [shortly](#), distributed workers do not require Gurobi licenses. You can add any machine to a Remote Services cluster to act as a distributed worker. The manager does require a distributed algorithm license (you'll see a `DISTRIBUTED=` line in your license file if distributed algorithms are enabled).

A typical distributed optimization will look like the following, with all machines belonging to the same Remote Services cluster:



The workload associated with managing distributed algorithms is quite light, so a machine can handle both the manager and worker roles without degrading performance.

Another option is to use a machine outside of your Remote Services cluster as the manager:



Note that we only allow a machine to act as manager for a single distributed job. If you want to run multiple distributed jobs simultaneously, you'll need multiple manager machines.

2.3 Roles

Users of Gurobi Remote Services will fall into one of three possible roles: [cluster administrator](#), [administrator](#), or [client](#). The cluster administrator is in charge of setting up the cluster, adding and removing nodes, etc. In contrast, administrators do things like monitor the length of the server queue, kill jobs, etc. Clients are the people that run the programs on client machines that ultimately submit jobs to a Compute Server cluster. Note that the client role and the administrator role are often, but not always, performed by the same person. The cluster administrator role is typically performed by a system administrator.

Gurobi includes a number of tools that are relevant to the people in these roles. These are all covered in much more detail later on, but we'll describe how they fit with the various roles in this section.

Note that a Gurobi Remote Services cluster will have three different passwords, corresponding to these different roles. You will need to provide the appropriate password for the type of Remote Services task you wish to perform. Again, the details will be discussed shortly.

Cluster Administrator

As you might expect, the cluster administrator manages a Remote Services cluster. The primary tool for doing so is [grb_rs](#), which is the program that runs on the Remote Services node and accepts requests from client programs. The cluster administrator will need to start this on all of the nodes of a Remote Services cluster.

A second important administrative tool is [grbcluster](#), which is used to issue commands to an already running cluster. Examples of cluster administrator commands include adding or removing nodes, and enabling or disabling job processing on a cluster. This tool provides a variety of commands, and is used for all three Remote Services roles. You can type `grbcluster --help` for a full list of commands.

For more details, please refer to the section about [setting up and administering a cluster](#).

Administrator

An administrator monitors and manages the flow of jobs through a Remote Services cluster. Examples of administrator commands include aborting jobs, changing cluster parameters and checking licenses. The primary tool for doing so is [grbcluster](#). You can get a full list of available commands by typing `grbcluster --help`.

Client

A Remote Services client submits jobs to the cluster. This is done through a user application or through the Gurobi command-line tool [gurobi_cl](#) (which is documented in the [Gurobi Command-Line Tool](#) section of the [Gurobi Reference Manual](#)). Submitting a job to a Remote Services cluster is typically just a matter of running the appropriate program. We'll provide a simple example in the next section.

Clients can also use [grbcluster](#) command to monitor the state of their jobs and of the Remote Services queue. Example commands include listing active jobs, listing recently executed jobs, displaying the log of a recent job, etc. You can get a full list of available commands by typing `grbcluster --help`.

2.4 Simple Example

After your cluster has been set up (which will be covered in [this section](#)), it is generally a simple matter to submit a job to it. As noted earlier, you have several options for indicating that you wish to offload your client Gurobi computation to a Compute Server: through command-line arguments, a license file, or a programming language API. The precise details will be covered in a [later section](#). For now, we'll simply demonstrate the use of command-line arguments using `gurobi_cl`.

The command `gurobi_cl stein9.mps` would solve the model stored in file `stein9.mps` on your local machine. By adding a command-line argument,

```
gurobi_cl --server=server1:61000 stein9.mps,
```

we can instead offload the computation to the Gurobi Remote Services cluster that is listening on port 61000 of machine `server1`:

```
> gurobi_cl --server=server1:61000 stein9.mps
Compute Server job ID: 1e9c304c-a5f2-4573-affa-ab924d992f7e
Capacity available on 'server1:61000' - connecting...
Established HTTP unencrypted connection

Gurobi Optimizer version 8.0.0 build v8.0.0rc0 (linux64)
Copyright (c) 2018, Gurobi Optimization, LLC

...

Optimal solution found (tolerance 1.00e-04)
Best objective 5.000000000000e+00, best bound 5.000000000000e+00, gap 0.0000%

Compute Server communication statistics:
  Sent: 0.0 MBytes in 34 msgs and 0.00s (0.00 MB/s)
  Received: 0.0 MBytes in 141 msgs and 0.00s (0.00 MB/s)
```

The initial log output indicates that a Compute Server job was created, that the Compute Server cluster had capacity available to run that job, and that an unencrypted HTTP connection was established with a server in that cluster. The log concludes with statistics about the communication performed between the client machine and the Compute Server.

We'll now move on to discussions of [setting up a cluster](#) and [using a cluster](#). The former is meant for cluster administrators. If you intend to be a client of a Remote Services cluster that has already been set up, you can skip this section.

Cluster Setup and Administration

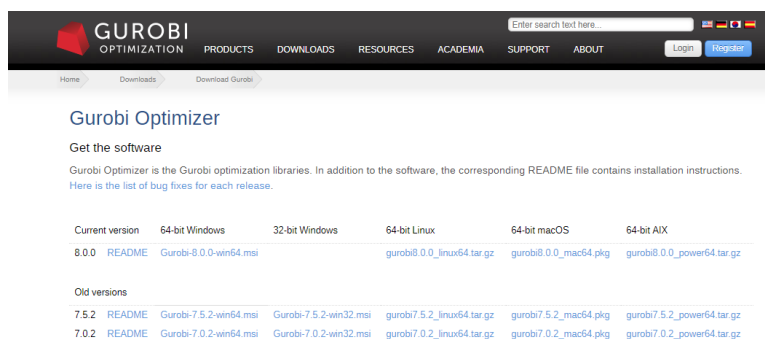
This section will cover the setup and administration of a Gurobi Remote Services cluster. The intended audience is the cluster administrator. If you are interested in using a cluster that has already been set up, you should proceed to the [next section](#).

3.1 Setting up Remote Services

The Gurobi Remote Services package must be installed on all the machines that will be part of the cluster. Your first step is to [download and install](#) Remote Services on all relevant server machines. Once installed, you will need to set up a [license](#), if necessary. Then, the [Remote Services agent](#) must be [configured](#) and started as a [standard process](#) or as a [service](#). Finally, you should [verify](#) your installation.

Installation

The first step in installing Gurobi Remote Services is to download the installer from [our download page](#). You'll need to find your platform and choose the corresponding file to download.



Make a note of the name and location of the downloaded file.

Your next step will depend on your platform:

Linux installation

On Linux, your next step is to choose a destination directory. We recommend `/opt` for a shared installation (you may need administrator privileges), but other directories will work as well. Copy the Remote Services distribution to the destination directory and extract the contents. Extraction is done with the following command:

```
tar xvfz gurobi_server8.0.0_linux64.tar.gz
```

This command will create a sub-directory `gurobi_server800/linux64` that contains the complete Linux Remote Services distribution. Assuming that you extracted the Gurobi server archive in the `/opt` directory, your `<installdir>` (which we'll refer to throughout this document) will be `/opt/gurobi_server800/linux64`.

The Gurobi Optimizer makes use of several executable files. In order to allow these files to be found when needed, you will have to modify your search path. Specifically, your `PATH` environment variable should be extended to include `<installdir>/bin`. Users of the `bash` shell should add the following line to their `.bashrc` file:

```
export PATH="${PATH}:/opt/gurobi_server800/linux64/bin"
```

Users of the `csh` shell should add the following line to their `.cshrc` file:

```
setenv PATH "${PATH}:/opt/gurobi_server800/linux64/bin"
```

You'll need to close your current terminal window and open a new one after you have made these changes in order to pick up the new settings.

In some Linux distributions, applications launched from the Linux desktop won't read `.bashrc` (or `.cshrc`). You may need to set the Gurobi environment variables in `.bash_profile` or `.profile` instead. Unfortunately, the details of where to set these variables vary widely among different Linux distributions. We suggest that you consult the documentation for your distribution if you run into trouble.

Mac OS Installation

On Mac OS, your next step once you've downloaded the Gurobi Remote Services package from our website is to double-click on the installer (e.g., `gurobi_server8.0.0_mac64.pkg` for Gurobi 8.0.0) and follow the prompts. By default, the installer will place the Gurobi Remote Services 8.0.0 files in `/Library/gurobi_server800/mac64` (note that this is the *system* `/Library` directory, not your personal `/Library` directory). Your `<installdir>` (which we'll refer to throughout this document) will be `/Library/gurobi_server800/mac64`.

Windows Installation

On Windows, your next step is to double-click on the Gurobi Remote Services installer that you downloaded from our website (e.g., `GurobiServer-8.0.0-win64.msi` for Gurobi 8.0.0).

Note: if you selected *Run* when downloading you've already run the installer and don't need to do it again.

By default, the installer will place the Gurobi 8.0.0 files in directory `c:/gurobi_server800/win64`. The installer gives you the option to change the installation target. We'll refer to the installation directory as `<installdir>`.

Licensing

If you are setting up a Gurobi Compute Server, you will need to download and install a license file (no license file is required if you only wish to use a machine as a distributed worker for distributed algorithms). You'll find detailed instructions for downloading a license in the *Retrieving and Setting Up a Gurobi License* section of the *Gurobi Optimizer Quick Start Guide*.

We'll just provide a quick summary of the process here. Your first step is to locate and download your license file from the [Gurobi License Center](#). When you download the license file, we strongly recommend that you place it in the default location:

- `C:\gurobi\` on Windows
- `/opt/gurobi/` on Linux
- `/Library/gurobi/` on Mac OS

You can also set the environment variable `GRB_LICENSE_FILE` to point to this file.

Remote Services Agent (`grb_rs`)

To form a Remote Services cluster, you need to run the *Remote Services agent* (`grb_rs`) on all the nodes that make up the cluster. These agents communicate amongst themselves, and also with client programs (e.g. `gurobi_cl`) and tools (e.g., `grbcluster`), to manage jobs, to provide information about the state of the cluster, etc.

The primary task of the Remote Services agents is to collectively manage the queueing and the execution of jobs. The agents work together to balance the load by assigning a new job to the node

with the fewest running jobs whenever possible. If all nodes are at capacity, newly submitted jobs will be queued, and the first node with available capacity will later execute the job. If a new node is added to the cluster, it will immediately start processing queued jobs.

The `grb_rs` executable provides several commands and flags to help in the configuration and execution of the agent. We will review these commands step by step in the following sections. You can get the full list of commands in the [reference section](#) or by using the command-line help:

```
> grb_rs --help
```

In the next sections, we will also use `grbcluster` to monitor and administrate the cluster. You can get the full list of commands in the [reference section](#) or by using the command-line help:

```
> grbcluster --help
```

Configuration

The Remote Services agent has a number of configuration properties that affect its behavior. These can be controlled using a `grb_rs.cnf` configuration file. By default, this file must be located in the same directory as the `grb_rs` executable. The installation package includes a predefined configuration file that can be used as a starting point (`<installdir>/bin/grb_rs.cnf`).

You can edit the default configuraton file or override it as `grb_rs` will use the following precedence rules:

- command line flag `--config`
- current directory
- shared directory (`C:\gurobi`, `/opt/gurobi`, `/Library/gurobi` depending on windows, linux and mac platforms respectively)
- directory where `grb_rs` is located

The configuration file contains a list of properties of the form `PROPERTY=value`. Lines that begin with the `#` symbol are treated as comments and are ignored. Here is an example:

```
# grb_rs.cnf configuration file
PASSWORD=abcd1234
ADMINPASSWORD=1234abcd
```

While you could create this file from scratch, we recommend you start with the version of this file that is included with the product and modify it instead.

Examples of properties that are configured through this file are client and administrator passwords, communication options, and job processing options. The command `grb_rs properties` lists all the available properties, the default values, and provides documentation for each. Some properties can be overridden on the command line of `grb_rs`; the name of the command-line flag you would use to do so is provided as well.

Some properties are important and must be changed for a production deployment:

HOSTNAME: This must be the DNS name of the node that can be resolved from the other nodes or the clients in your network. `grb_rs` tries to get a reasonable default value, but this value may still not be resolved by clients and could generate connection errors. In this case, you need to override this name in the configuration file with a fully qualified name of your node, for example:

HOSTNAME=server1

If the names cannot be resolved by clients, another option is to use IP addresses directly, in this case set this property to the IP address of the node.

CLUSTER_TOKEN: The token is a private key that enables different nodes to join the same cluster. All nodes of a cluster must have the same token. We recommended that you generate a brand new token when you set up your cluster. The **grb_rs token** command will generate a random token, which you can copy into the configuration file.

PASSWORD: This is the password that clients must supply in order to access the cluster. It can be stored in clear text or hashed. We recommended that you create your own password, and that you store it in hashed form. You can use the **grb_rs hash** command to compute the hashed value for your chosen password.

```
grb_rs hash newpass
$$ppEieKZExlBR-pCSUMlmc4oWlG8nZsU0E2IM0hJbzsmV_Yjj
```

Then copy and paste the value in the configuration file:

```
PASSWORD=$$ppEieKZExlBR-pCSUMlmc4oWlG8nZsU0E2IM0hJbzsmV_Yjj
```

The default password is **pass**.

ADMINPASSWORD: This is the password that clients must supply in order to run restricted administrative job commands. It can be stored in clear text or hashed. We recommended that you create your own password, and that you store it in hashed form. You can use the **grb_rs hash** command to compute the hashed value for your chosen password. The default password is **admin**.

CLUSTER_ADMINPASSWORD: This is the password that clients must supply in order to run restricted administrative cluster commands. It can be stored in clear text or hashed. We recommended that you create your own password, and that you store it in hashed form. You can use the **grb_rs hash** command to compute the hashed value for your chosen password. The default password is **cluster**.

JOBLIMIT: This property sets the maximum number of jobs that can run concurrently when using Compute Server on a specific node. The limit can be changed on a running cluster using the **grbcluster config** command, in which case the new value will persist and the value in the configuration file will be ignored from that point on (even if you stop and restart the cluster).

HARDJOBLIMIT: Certain jobs (those with priority 100) are allowed to ignore the JOBLIMIT, but they aren't allowed to ignore this limit. Client requests beyond this limit are queued. This limit is set to 0 by default which means that it is disabled.

CLIENT_DETAILS_ADMIN: When a job is submitted, the client hostname, IP, and process ID are recorded. By default, this information is displayed to any user running the command line tool **grbcluster** or the REST API. If this property is set to **true**, only the administrator will be able to access this information.

USERNAME_ADMIN: When a job is submitted, the client process username is recorded. By default, this information is displayed to any user running the command line tool **grbcluster** or the REST API. If this property is set to **true**, only the administrator will be able to access this information.

The configuration file is only read once, when **grb_rs** first starts. Subsequent changes to the file won't affect parameter values on a running server.

Starting Remote Services as a Process

Once you've installed the Remote Services package (including retrieving and installing your license file and, for Linux users, setting your **PATH** variable), starting **grb_rs** as a standard process is quite straightforward. From a terminal window with administrator privileges, simply issue the following command:

```
> grb_rs
```

This will start Remote Services agent on the default port (port 80).
You should see output like the following...

```
info : Reading config file: /home/jones/gurobi_server800/linux64/bin/grb_rs.cnf
info : Gurobi Remote Services starting...
info : Platform is linux
info : Version is 8.0.0 (build 128)
info : Node address is server1
info : Accepting worker registration on port 41173...
info : API server, started (HTTP) on port 80...
```

If you do not have administrator privileges or if the default port is already in use, you will see an error about opening the port. For example, on Linux you might see an error like this:

```
fatal : Gurobi Remote Services terminated, listen tcp :80: bind: permission denied
```

or

```
fatal : Gurobi Remote Services terminated, listen tcp :80: bind: address already in use
```

Note that **grb_rs** does not have to be run with elevated privileges, but it does need elevated privileges to use the default port 80.

If you'd like to run **grb_rs** on a non-default port, use the **--port** flag or set the **PORT** property in the configuration file. For example:

```
> grb_rs --port=61000
```

The Remote Services agent (**grb_rs**) needs a directory to store various files, including the runtimes, job metadata, job log files, etc. The default location is a directory named **data**, located in the same directory as the **grb_rs** executable (**<installdir>/bin/data**). If you have a **data** directory in your current directory, it will be taken first.

If starting **grb_rs** produces an error message that indicates that there was a problem creating the storage service (as shown below), a likely cause is that another **grb_rs** process is already running.

```
fatal : Error creating storage service: Error opening data store: timeout
```

If you wish to start multiple `grb_rs` processes on the same machine for testing purposes (this is not recommended for production use), you will need to make sure each instance of `grb_rs` is started on a different port and using a different data directory. The command `grb_rs init` will help you by copying the default configuration and the data directory into a current directory.

For example, to start two nodes on the same machine with a hostname of `myserver`:

1. In a first terminal window, create a new directory `node1`,
2. Change your current directory to `node1` and run `grb_rs init`
3. Start the first node:

```
grb_rs --port=61000
```

4. In a second terminal window, create a new directory `node2`,
5. Change your current directory to `node2` and run `grb_rs init`
6. Start the second node on a different port and join the first node:

```
grb_rs --port=61001 --join=myserver:61000
```

Starting Remote Services as a Service

While you always have the option of running `grb_rs` from a terminal and leaving the process running in the background, we recommended that you start it as a service instead, especially in a production deployment. The advantage of a service is that it will automatically restart itself if the computer is restarted or if the process terminates unexpectedly.

`grb_rs` provides several commands that help you to set it up as a service. These must be executed with administrator privileges:

grb_rs install: Install the service. The details of exactly what this involves depend on the host operating system type and version: this uses `systemd` or `upstart` on Linux, `launchd` on MacOS, and Windows services on Windows.

grb_rs start: Start the service (and install it if it hasn't already been installed).

grb_rs stop: Stop the service.

grb_rs restart: Stop and then start the service.

grb_rs uninstall: Uninstall the service.

Note that the `install` command installs the service using default settings. If you don't need to modify any of these, you can use the `start` command to both install and start the service. Otherwise, run `install` to register the service, then modify the configuration (the details are platform dependent and are touched on below), and then run `start` the service.

Note that you only need to start the service once; `grb_rs` will keep running until you execute the `grb_rs stop` command. In particular, it will start again automatically if you restart the machine.

Note also that the `start` command does not take any flags or additional parameters, and that all the configuration properties must be set in the `grb_rs.cnf` configuration file. If you need to

make a change to the configuration, use the command **stop** then the command **start** in order to restart **grb_rs** with the updated configuration. The one exception is the **JOBLIMIT** property, which can be changed on a live server using **grbcluster**. If you change this property and restart the server, the new value will persist and the value in the configuration file will be ignored.

The exact behavior of these commands varies depending on the host operating system and version.

Linux

On Linux, **grb_rs** supports two major service managers **systemd** and **upstart**. The **install** command will detect the service manager available on your system and will generate a service configuration file located in **/etc/systemd/system/grb_rs.service** or **/etc/init/grb_rs.conf** for **systemd** and **upstart**, respectively. Once the file is generated, you can edit it to set advanced properties. Please refer to the documentation of **systemd** or **upstart** to learn more about service configuration.

Use the **start** and **stop** commands to start and stop the service. When the service is running, the log messages are sent to the Linux syslog and to a rotating log file, **service.log**, located in the same directory as **grb_rs**.

The **uninstall** command will delete the generated file.

Mac OS

On Mac OS, the system manager is called **launchd**, and the **install** command will generate a service file in **/Library/LaunchDaemons/grb_rs.plist**. Once the file is generated, you can edit it to set advanced properties. Please refer to the **launchd** documentation to learn more about service configuration.

Use the **start** and **stop** commands to start and stop the service. When the service is running, the log messages are sent to the Mac OS syslog and to a rotating log file, **service.log**, located in the same directory as **grb_rs**.

The **uninstall** command will delete the generated file.

Windows

On Windows, the **install** command will declare the service to the operating system. If you wish to set advanced properties for the service configuration, you will need to start the **Services** configuration application. Please refer to the Windows Operating System documentation for more details.

Use the **start** and **stop** commands to start and stop the service. When the service is running, the log messages are sent to the Windows event log and to a rotating log file, **service.log**, located in the same directory as **grb_rs**.

The **uninstall** command will delete the service from the registry.

Verification

Once you have **grb_rs** running, you can check to make sure that you will be able to submit jobs to it by issuing the following command from any machine that can reach the server on your network:

```
> grbcluster nodes --server=server1 --password=pass --long
ADDRESS STATUS TYPE    LICENSE PROCESSING #Q #R JL IDLE %MEM  %CPU  STARTED          RUNTIMES VERSION
server1 ALIVE  COMPUTE VALID    ACCEPTING  0  0  1  43m  42.67  2.53  2017-09-13 20:17:10 [8.0.0]  8.0.0
```

You are ready to submit jobs if both of the following are true:

- the **STATUS** column indicates that one or more servers are **ALIVE**
- the **LICENSE** column indicates that the license is **VALID** (or **N/A** for distributed workers).

If **grbcluster** is unable to connect or if it does not show any live nodes, then check your network and the log of the **grb_rs** nodes (the console output or `<installdir>/bin/service.log` if started as a service).

If a node has an **INVALID** license, the **ERROR** field will provide more information about the error. For example:

```
> grbcluster licenses --server=server1 --password=admin
ADDRESS STATUS  TYPE KEY EXP ORG USER APP VER CS   DL  ERROR
server1 INVALID NODE                                false 0   No Gurobi license found...
```

Optionally, you can also check that submitting a job is successful. To this end, you may want to identify a machine from which the users will typically submit jobs and install the gurobi client package. Then, you can submit a job running the following command:

```
> gurobi_cl --server=server1 --password=pass misc07.mps
```

For more information on how to install the client and run **gurobi_cl** please refer to the section about [using Remote Services](#).

Note that if you started the node with a specific port, you can specify it in the server URL:

```
> grbcluster nodes --server=server1:61000 --password=pass
...
> gurobi_cl --server=server1:61000 --password=pass misc07.mps
```

3.2 Forming a Cluster

As noted earlier, a cluster consists of a set of one or more nodes, all running `grb_rs`. This section explains how to [form a cluster](#). Multi-node clusters provide additional capabilities relative to single-node clusters. For Compute Server, a multi-node cluster will automatically balance computational load among the various member nodes. For distributed algorithms, a multi-node cluster enables various algorithms to distribute work among multiple machines. This section begins by discussing the different [types of nodes](#) that are needed to support both Compute Server and distributed algorithms. Next, we will explain the [grouping](#) feature that can be used to create subsets of nodes to process some jobs. Finally, we will discuss the dynamic nature of a cluster. The nodes in a cluster can independently [start and stop processing jobs](#), either to simplify maintenance or to scale the processing capacity up or down.

Connecting Nodes

Every Remote Services cluster starts with a single node. The steps for starting Remote Services on a single node, either as a [standard process](#) or as a [service](#), were covered in earlier sections.

Before adding nodes into your cluster, you first need to make sure that the cluster token (property `CLUSTER_TOKEN` in the configuration file) has the same value in each node. For better security, we recommend that you change the predefined value of the token by generating a new one and pasting the same value into each node configuration file. You can generate a new token with the following command:

```
> grb_rs token
GRBTK-6o4xuj59WJ05508nmaNwc1TtjZJAL1UcwN4vTD4qK4nata8oLr9GnubyXrLTkggc/aw2A==
```

Similarly, the passwords used for client, administrator and cluster administrator must be the same in all nodes. For better security, it is recommended to change the predefined value of the passwords by choosing a new password, generating a hash value for that password, and then pasting the result into each node configuration file. You can generate a hash of your chosen password (e.g., `mynewpass`) with the following command:

```
> grb_rs hash mynewpass
$$v0UBWkM_9kpY_v2RECV2LBGnlr8qzaGHzf0fMJvrMYwPnJap
```

Adding nodes to your cluster

Once you've started a single-node cluster, you can add nodes using the `--join` flag to `grb_rs` or the `JOIN` configuration property. For example, if you've already started a cluster on the default port of `server1`, you would run the following command on the new node (call it `server2`) to create a two-node cluster:

```
> grb_rs --join=server1
```

In the log output for `server2`, you should see the result of the handshake between the servers:

```
info : Node server1, transition from JOINING to ALIVE
```

Similarly, the log output of `server1` will include the line:

```
info : Node server2, added to the cluster
```

If you are using a non-default port, you can specify the target node port as part of the node URL in the `--join` flag and you can specify the port of the current node using the `--port` flag. You could use different ports on the different machines, but it is a good practice to use the same one, for example 61000. The command would look like this instead:

```
> grb_rs --join=server1:61000 --port=61000
```

The JOIN property can also be set through the configuration file in the same way:

```
JOIN=server1:61000
PORT=61000
```

When starting **grb_rs** as a service, you won't have the opportunity to provide command-line options, so you'll need to provide this information through the configuration file.

Once you've created a multi-node cluster, you can add additional nodes to that cluster by doing a JOIN using the name of any member node. Furthermore, the **--join** flag or the JOIN property can take a comma-separated list of node names, so a node can still join a cluster even if one of the member nodes is unavailable. Note that when a list of nodes is specified, the joining node will try to join all the specified nodes at the same time. Joining nodes is an asynchronous process, if some target nodes are not reachable, the joining node will retry before giving up on joining. If all the nodes are reachable, they will all join and form a single cluster.

Checking the status of your cluster

Using **grbcluster**, you can check the status of the cluster:

```
> grbcluster --server=server1 --password=pass nodes --long
ADDRESS STATUS TYPE    LICENSE PROCESSING #Q #R JL IDLE      %MEM  %CPU  STARTED          RUNTIMES VERSION
server1 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h59m0s  9.79  0.50  2017-09-27 17:03:24 [8.0.0]  8.0.0
server2 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h46m0s  8.75  0.00  2017-09-27 17:16:11 [8.0.0]  8.0.0
```

The nodes of the cluster are constantly sharing information about their status. When using **grbcluster**, you can use any of the nodes in the **--server** flag for all global commands.

Each node can be in one of the following states:

ALIVE: The node is up and running.

DEGRADED: The node failed to respond to recent communications. The node could return to the **ALIVE** state if it can be reached again. The node will stay in this state until a timeout (controlled by the configuration property **DEGRADED_TIMEOUT**), at which point it is considered as **FAILED**

FAILED: The node has been in **DEGRADED** state for too long, and has been flagged as **FAILED**. A node will remain in the **FAILED** state for a short time, and it will eventually be removed from the cluster. If the node comes back online, it will not re-join the cluster automatically.

JOINING: The node is in the process of joining the cluster.

LEAVING: The node left the cluster. It will stay in that state for short time period and then it will be removed from the cluster.

You can dynamically add or remove a node from a cluster using the **grbcluster join** or **grbcluster leave** commands. The **join** command can be useful when you want a node to rejoin the cluster after a network issue without having to restart the node. For example, if **server2** left the cluster after a failure, it could rejoin using the following command:

```
> grbcluster --server=server2 --password=cluster join server1
```

Compute Servers and Distributed Workers

A Remote Services cluster is a collection of nodes of two different types:

COMPUTE: A Compute Server node supports the offloading of optimization jobs. Features include load balancing, queueing and concurrent execution of jobs. A Compute Server license is required on the node. A Compute Server node can also act as a distributed worker.

WORKER: A distributed worker node can be used to execute part of a distributed algorithm. A license is not necessary to run a distributed worker, because it is always used in conjunction with a manager (another node or a client program) that requires a license. A distributed worker node can only be used by one manager at a time (i.e., the job limit is always set to 1).

By default, `grb_rs` will try to start a node in Compute Server mode and the node license status will be `INVALID` if no license is found. In order to start a distributed worker, you need to set the `WORKER` property in the `grb_rs.cnf` configuration file (or the `--worker` command-line flag):

```
WORKER=true
```

Once you form your cluster, the node type will be displayed in the `TYPE` column of the output of `grbcluster nodes`:

```
> grbcluster --server=server1 --password=pass nodes --long
```

ADDRESS	STATUS	TYPE	LICENSE	PROCESSING	#Q	#R	JL	IDLE	%MEM	%CPU	STARTED		RUNTIMES	VERSION
server1	ALIVE	COMPUTE	VALID	ACCEPTING	0	0	2	46h59m0s	9.79	0.50	2017-09-27 17:03:24	[8.0.0]	8.0.0	
server2	ALIVE	COMPUTE	VALID	ACCEPTING	0	0	2	46h46m0s	8.75	0.00	2017-09-27 17:16:11	[8.0.0]	8.0.0	
server3	ALIVE	WORKER	N/A	ACCEPTING	0	0	1	46h46m0s	8.75	0.00	2017-09-27 17:16:11	[8.0.0]	8.0.0	
server4	ALIVE	WORKER	N/A	ACCEPTING	0	0	1	46h46m0s	8.75	0.00	2017-09-27 17:16:11	[8.0.0]	8.0.0	

The node type cannot be changed once `grb_rs` has started. If you wish to change the node type, you need to stop the node, change the configuration, and restart the node. You may have to update your license as well.

Distributed Optimization

When using distributed optimization, distributed workers are controlled by a manager. There are two ways to set up the manager:

- The manager can be a job running on a Compute Server. In this case, the manager job is first submitted to the cluster and executes on one of the `COMPUTE` nodes as usual. When this job starts, it will also request some number of workers (see parameters `DistributedMIPJobs`, `ConcurrentJobs`, or `TuneJobs`). The first choice will be `WORKER` nodes. If not enough are available, it will use `COMPUTE` nodes. The workload associated with managing the distributed algorithm is quite light, so the initial job will act as both the manager and the first worker.
- The manager can be the client program itself. The manager does not participate in the distributed optimization. It simply coordinates the efforts of the distributed workers. The manager will request distributed workers (using the `WorkerPool` parameter), and the cluster will first select the `WORKER` nodes then, if not enough are available, it will use `COMPUTE` nodes as well.

In both cases, the machine where the manager runs must be licensed to run distributed algorithms (you should see a `DISTRIBUTED=` line in your license file).

It is typically better to use the Compute Server itself as the distributed manager, rather than the client machine. This is particularly true if the Compute Server and the workers are physically close to each other, but physically distant from the client machine. In a typical environment, the client machine will offload the Gurobi computations onto the Compute Server, and the Compute Server will then act as the manager for the distributed computation.

Grouping

With the Remote Services grouping feature, you can define a subset of the nodes in your cluster as a group, and then submit jobs specifically to that group. This can be quite useful when some nodes in the cluster are different from others. For example, some nodes may have more memory or faster CPUs. Using this feature, you can force jobs to only run on the appropriate type of machines. If all nodes of the requested group are at capacity, jobs will be queued until a member of that group is available.

In order to define a group, you will need to add the `GROUP` property to the `grb_rs.cnf` configuration file and give a name to the group:

```
GROUP=group1
```

The groups are static and can only be changed in the node configuration file. If you wish to change the group of a node, you will need to stop the node, edit the configuration, and restart the node. A node can only be a member of one group.

The `grbcluster nodes` command displays the assigned group for each node (in the `GRP` column):

```
> grbcluster --server=server1 --password=pass nodes
ADDRESS STATUS TYPE   GRP   LICENSE #Q #R JL IDLE   %MEM %CPU
server1 ALIVE  COMPUTE group1 VALID   0 0 2 46h59m0s 9.79 0.50
server2 ALIVE  COMPUTE group1 VALID   0 0 2 46h46m0s 8.75 0.00
server3 ALIVE  COMPUTE          VALID   0 0 2 46h46m0s 8.75 0.00
server4 ALIVE  COMPUTE          VALID   0 0 2 46h46m0s 8.75 0.00
```

With `gurobi_cl`, you can submit a job to a given group by using the `GROUP` property of the client license file (see [set up a client license](#)).

Processing State and Scaling

Each node of the cluster can be in one of three processing states:

ACCEPTING: The node is accepting new jobs.

DRAINING: The node is not accepting new jobs, but it is still processing existing jobs.

STOPPED: The node is not accepting new jobs and no jobs are running.

A node always starts in the **ACCEPTING** state. If you need to perform maintenance on a node, or if you want the node to leave the cluster in a clean way for other reasons, the cluster administrator can issue the `stop` command:

```
> grbcluster --server=server1 --password=cluster stop
```

If jobs are currently running, the state will change to **DRAINING** until the jobs finish, at which point it will change to **STOPPED**. If no jobs are running, the state will change to **STOPPED** immediately. In the **DRAINING** or **STOPPED** states, new jobs are rejected on that node. However, the node is still a member of the cluster and all the other features, commands, and APIs are still active.

Once a node is in the **STOPPED** state, you can safely remove it from the cluster (to perform maintenance, shut it down, etc.). To return it to the cluster and resume job processing, run the **start** command:

```
> grbcluster --server=server1 --password=cluster start
```

The flag **--server** is used to target a specific node in the cluster. Adding the **--all** flag requests that the command (e.g., **start** or **stop**) be applied to all nodes in the cluster.

By using the **start** and **stop** with a cluster of Compute Servers, you can effectively scale your cluster up or down, depending on the current cluster workload.

- You can scale down the cluster by stopping the processing on some nodes.
- You can scale up the cluster by starting new nodes or resuming processing on some nodes. As soon as a node starts or resumes processing, it will pick up jobs from the current queue or wait for new jobs to be submitted.

3.3 Communication Options

A node running Gurobi Remote Services communicates with clients through a REST API using HTTP by default. For more secure deployments, [HTTPS can be enabled](#). Remote Services also support [self-signed certificates](#) for testing your deployment. Finally, [firewalls](#) may have to be configured to open the port used by the cluster and for advanced networking setup, a remote services [router](#) can be used.

Using HTTPS

Several properties can be used to configure the communication options. In order to enable HTTPS with TLS data encryption over the wire, you need to set the TLS property.

```
TLS=true
```

You will also need to provide the paths to the private key and the certificate files:

```
TLS_CERT=cert.pem
```

```
TLS_KEY=key.pem
```

When HTTPS is enabled, the standard HTTPS port 443 is then used as the default instead of port 80. As with the port 80, you will need to start `grb_rs` with elevated privileges. Otherwise, you will get a permission error. On Linux, you'd see an error message like the following:

```
fatal : Gurobi Remote Services terminated, listen tcp :443: bind: permission denied
```

As explained in the installation section, you can change the port using the `PORT` property. Note that you cannot mix nodes using HTTP and nodes using HTTPS in the same cluster. If you wish to use HTTPS, all the nodes must be configured in the same way. HTTPS will be used for communication between the nodes and also between the clients and the nodes.

If you enable HTTPS, you will need to specify the prefix `https://` when accessing any nodes of the cluster:

```
> grbcluster --server=https://server1 --password=pass nodes
ADDRESS      STATUS TYPE    LICENSE #Q #R JL IDLE   %MEM %CPU
https://server1 ALIVE  COMPUTE VALID   0 0 2 46h59m 9.79 0.50
https://server2 ALIVE  COMPUTE VALID   0 0 2 46h46m 8.75 0.00
```

Using HTTPS with Self-Signed Certificates

Using self-signed certificates is not recommended for production deployment as it is less secure, but it can be useful when testing a deployment. If you do not specify a key and a certificate in the `TLS_KEY` and `TLS_CERT` properties, `grb_rs` will generate them for you at startup. You can also specify your own self-signed certificate using `TLS_KEY` and `TLS_CERT` properties.

To use a self-signed certificate, you'll need to activate insecure mode by setting the `TLS_INSECURE` property:

```
TLS_INSECURE=true
```

When using this mode, the data will be encrypted over the wire, and the default port will be 443, but the certificate will not be validated.

On the client side, you will also need to activate this mode either by using the `--tls-insecure` flag or by setting the `GRB_TLS_INSECURE` environment variable:

```
> grbcluster --tls-insecure --server=https://server1 --password=pass nodes
ADDRESS      STATUS TYPE    LICENSE #Q #R JL IDLE   %MEM %CPU
https://server1 ALIVE  COMPUTE VALID   0 0 2 46h59m 9.79 0.50
https://server2 ALIVE  COMPUTE VALID   0 0 2 46h46m 8.75 0.00
```

Firewalls

As noted earlier, a node running Gurobi Remote Services communicates with clients through a REST protocol over HTTP or HTTPS using a single port. It uses standard port 80 for HTTP or 443 for HTTPS by default, but you can choose an arbitrary port through the `PORT` configuration property. If there is a firewall between the clients and the nodes of the cluster, the chosen port will have to be open.

The command line tools and the libraries are also compatible with standard proxy settings using environment variables `HTTP_PROXY` and `HTTPS_PROXY`. `HTTPS_PROXY` takes precedence over `HTTP_PROXY` for https requests. The values may be either a complete URL or a `host[:port]`, in which case the `http` scheme is assumed.

If you face connectivity issues with firewalls or proxy servers, we suggest you share this section with your network administrator.

Router

A Remote Services Router may be used when you need to isolate better the cluster from the clients. Without a router, the clients need to have direct access to each node in the cluster and the node DNS name and IP address must be accessible from the clients.

Instead, a router provides a point of contact for all clients and will route the communication to the appropriate node in the cluster. A Remote Services Router acts as a reverse proxy. Behind a router, the cluster nodes can use private DNS names or IP addresses as long as all the nodes and the router can communicate together. Only the router must be accessible from the clients.

In addition, the router will use HTTP as default and can also use HTTPS so that the data can be encrypted from the clients to the router. The router can then route the traffic using HTTPS or HTTP depending on the configuration of the cluster. It is a common configuration to enable HTTPS only between the clients and the router while having the router and the nodes communicate over unencrypted HTTP in a private network. Using this setup you only have to manage certificates on the router.

You can get more information about the router (`grb_rsr`) by reading the command line help:

```
grb_rsr --help
```

The router uses a configuration file `grb_rsr.cnf` that must be placed in the same directory as `grb_rsr` executable. A predefined configuration file with additional comments is provided. The following commands lists the available configuration properties:

```
grb_rsr properties
```

In a similar way to `grb_rs`, the router can be started as a service and the log messages will be stored in the `grbrsr-service.log` rotating file by default. The log messages will also be sent to the syslog on mac and linux, and to the service event log on Windows.

```
grb_rsr start
```

We will refer to the router URL as the full URL to access the router over HTTP or HTTPS and using standard port or a custom one. Here are some examples:

```
http://router.mycompany.com
http://router.mycompany.com:61001
https://router.mycompany.com
https://router.mycompany.com:61001
```

When using the command line tools `grbcluster` or `gurobi_cl`, you can specify the router URL using the `--router` flag. You can also add the property `ROUTER` to your license file. For example, once you have configured and started the router and your cluster, you can display the cluster status with the following command:

```
> grbcluster --router=http://router.mycompany.com --server=server1 ---password=pass nodes
ADDRESS STATUS TYPE    LICENSE #Q #R JL IDLE  %MEM  %CPU
server1 ALIVE  COMPUTE VALID    0  0  2  46h59m 9.79  0.50
server2 ALIVE  COMPUTE VALID    0  0  2  46h46m 8.75  0.00
```

For the clients using the Gurobi Optimizer API, you will need to either set the `ROUTER` property in the license file or construct an empty environment and set the `CSRouter` parameter before starting the environment.

For clients using the cluster REST API for monitoring purpose, you will need to use the router URL instead of a node address, and you can pass the selected node address in the header `X-GUROBI-SERVER`. This way, the client communicates with the router and the router will use the header value to forward the request to the selected node. In case the node address is incorrect or does not exist, the router will return the HTTP error code 502.

3.4 Maintaining a Cluster

To expand a bit on our earlier description, a Remote Services cluster consists of a set of one or more nodes running the Remote Services agent, as well as a set of runtimes that enable those nodes to execute optimization jobs. You can think of a runtime as a specific version of the Gurobi Optimizer that solves the optimization problems that are offloaded to a node in the cluster. In this section, we will first explain how to [update and manage the runtimes](#). We will then explain how to [upgrade the cluster](#) to a new version of the Remote Services and runtimes.

Managing Runtimes

A runtime is an executable built to run jobs using a given version of the Gurobi Optimizer. Each node in the cluster can handle multiple runtimes, so different Gurobi versions can be supported on the same node at the same time. The Gurobi Remote Services agent will automatically select the appropriate runtime, depending on the version of the Gurobi Optimizer library used by the client program.

Runtime executables, named `grb_rsw`, are installed in the `data` directory of a node, under the following directory structure (the version numbers used in this example are just for demonstration purpose):

```
grb_rs
data/
  runtimes/
    v8.0.0/
      grb_rsw
    v8.0.1/
      grb_rsw
```

The Remote Services agent will select the latest technical release that matches the major and minor version of the client. With the example above, if the client uses version 8.0.0, runtime version 8.0.1 will be selected. If later a version 8.0.2 is installed, the same client will use it without any modification.

Note that the Remote Services agent should be no older than the runtimes you deploy for it. Thus, for example, you can deploy runtime version 8.0.0 in an agent version 8.5, but not vice-versa. Note also that the ability to support different versions in a single Compute Server only started with version 8.0, so older versions such as 7.0 or 7.5 are not supported.

Deploying Runtimes

The Remote Services installation package will contain the latest supported runtime, which will be ready to use. You don't need to take any action to install a runtime when you first install Remote Services.

When new versions are released, you have the choice of reinstalling Remote Services with the latest runtimes or deploying only the runtimes that you need.

To deploy a specific runtime to a running node, you first need to stop the processing on that node:

```
grbcluster --server=server1 --password=cluster stop
```

Once all running jobs have finished processing and the node processing state has changed to STOPPED, you can deploy the new runtime using the `grbcluster deploy` command:

```
>grbcluster --server=server1 --password=cluster deploy gurobi_server801/linux64/bin/data/runtimes/v8.0.1/grb_rsw
```

You can examine the list of available runtimes using the `grbcluster nodes` command. Available versions are listed in the `RUNTIMES` column:

```
> grbcluster --server=server1 --password=pass nodes --long
ADDRESS STATUS TYPE    LICENSE PROCESSING #Q #R JL IDLE  %MEM %CPU STARTED          RUNTIMES    VERSION
server1 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h59m 9.79  0.50 2017-09-27 17:03:24 [8.0.0,8.0.1] 8.0.0
server2 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h46m 8.75  0.00 2017-09-27 17:16:11 [8.0.0]        8.0.0
```

You can remove an old runtime using `grbcluster undeploy`:

```
> grbcluster --server=server1 --password=cluster undeploy 8.0.0

> grbcluster --server=server1 --password=pass nodes --long
ADDRESS STATUS TYPE    LICENSE PROCESSING #Q #R JL IDLE  %MEM %CPU STARTED          RUNTIMES    VERSION
server1 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h59m 9.79  0.50 2017-09-27 17:03:24 [8.0.1]      8.0.0
server2 ALIVE  COMPUTE VALID    ACCEPTING  0  0  2  46h46m 8.75  0.00 2017-09-27 17:16:11 [8.0.0]      8.0.0
```

After deploying or undeploying, you can resume the job processing on the node, at which point new jobs will use the latest runtimes:

```
> grbcluster --server=server1 --password=cluster start
```

You can use the flag `--all-stopped` with the `deploy` or `undeploy` commands to deploy or undeploy to multiple nodes at a time. Note that this flag will only apply to nodes that are already `STOPPED`, so you should issue the `stop` command first (typically with the `--all` flag) to stop the nodes.

In conclusion, you can incrementally deploy new runtimes on cluster nodes as they become available without having to reinstall Remote Services. This works only for technical releases, and if you do not need to deploy a fix in the Remote Services stack.

Upgrading Remote Services

While you can deploy new technical releases within an existing Remote Services installation, a new major or minor release will require a Remote Services upgrade. The upgrade must be done on all nodes in your cluster. The steps are as follows:

1. Stop all nodes in your cluster.
2. Install the new Remote Services package on all nodes, and adjust the configuration file (`grb_rs.cnf`) to reflect any changes you made previously. If you started `grb_rs` as a service, you will need to uninstall the service first, and then install it again.
3. If necessary, upgrade your license file (or modify `GRB_LICENSE_FILE` to point to the new license file).
4. Start and join all the nodes to re-form the cluster.

See the earlier section on [setting up Remote Services](#) for details on installing the Remote Services package, and the section on [processing state](#) for details on starting and stopping nodes.

The Gurobi Compute Server feature was designed to be almost entirely transparent to both the developers and the users of the programs that use it. However, there are a few topics that you may need to be aware of, including [setting up a Compute Server client](#), [setting job priorities](#), [performance considerations](#), [callbacks](#), and a few [coding practices for Compute Server](#).

4.1 Client Configuration

Client configuration is generally quite straightforward. Assuming you've installed the Gurobi Optimizer on your client machine already, the main remaining tasks are to [set up a client license](#) and possibly adjust [job priorities](#), which affects the order in which jobs are processed. This section presents further information.

License File

A client program will always need to be told how to reach the Remote Services cluster. There are generally two ways to do this. The first is through the programming language APIs. We'll discuss this option in a later section on [programming with Remote Services](#). The second is through a license file. You can create a client license file yourself or edit an existing one, using your favorite text editor (Notepad is a good choice on Windows). The license file should be named `gurobi.lic`.

The license file contains a list of properties of the form `PROPERTY=value`. Lines that begin with the `#` symbol are treated as comments and are ignored. The license file must be placed in your home directory or in one of the following locations:

- `C:\gurobi\` on Windows
- `/opt/gurobi/` on Linux
- `/Library/gurobi/` on Mac OS

You can also set the environment variable `GRB_LICENSE_FILE` to point to this file.

Here are the properties you can set:

COMPUTESERVER: The fully qualified name of the main node used to access the cluster, plus the protocol scheme and port (if needed). For example, use `server1` to access a cluster using HTTP on the default port, or `https://server1:61000` to access a cluster over HTTPS using port 61000. You can also specify a comma-separated list of names so that other nodes can be used in case the first node can't be reached.

PASSWORD: The client password to access the cluster.

PRIORITY: Job Priority. Higher priority jobs take precedence over lower priority jobs. Priorities will be discussed in more detail shortly.

GROUP: Job group. If your cluster has been set up with groups, you can specify the group to submit the job to. The job will only be executed on nodes that are members of this group.

QUEUETIMEOUT: Queuing timeout (in seconds). A job that has been sitting in the queue for longer than the specified `QUEUETIMEOUT` value will return with a `JOB_REJECTED` error.

IDLETIMEOUT: Idle job timeout (in seconds). This property allows you to set a limit on how long a Compute Server job can sit idle before the server kills the job.

Here is a complete example:

```
COMPUTESERVER=server1:61000
PASSWORD=abcd
PRIORITY=10
```

The `gurobi_cl` or `grbcluster` tools provide command-line flags that allow you to set most of these properties. These tools will read the license file, but values specified via these command-line flags will override any values provided in the license file.

Queueing, Load Balancing, and Job Priorities

As noted earlier, Gurobi Compute Servers support job priorities. You can assign an integer priority between -100 and 100 to each job (the default is 0). When choosing among queued jobs, the Compute Server will run the highest priority job first. Note that servers will never preempt running jobs.

We have chosen to give priority 100 a special meaning. A priority 100 job will start immediately, even if this means that a server will exceed its job limit. You should be cautious with priority 100 jobs, since submitting too many at once could lead to very high server loads, which could lead to poor performance and even crashes in extreme cases. Note that this feature must be enabled by the cluster administrator using the `HARDJOBLIMIT` configuration property.

As with most job properties, priorities can be set either through a programming language API or through the license file.

Submitting Jobs with `gurobi_cl`

The Gurobi command-line tool can be used to submit optimization jobs to a Compute Server. The following gives a simple example:

```
> gurobi_cl --server=server1 --password=password1 misc07.mps
```

Note that we've included connection information for the Compute Server (in this case, the server name is `server1`, we're using the default port 80, and the password is `password1`). You could also provide connection information in a license file:

```
COMPUTESERVER=server1
PASSWORD=password1
```

Assuming the license file is found by `gurobi_cl`, you would then be able to omit the connection information:

```
> gurobi_cl misc07.mps
```

Issuing Cluster Management Commands: Using `grbcluster`

Your primary tool for issuing cluster management commands is a command-line program called `grbcluster`. The format of cluster management commands is:

```
grbcluster command [--flags]*
```

You can also ask for general help:

```
grbcluster --help
```

or help with a specific command:

```
grbcluster command --help
```

Cluster management commands can be run from any machine that can access the server using HTTP or HTTPS. You'll need the client password for your Compute Server cluster for most of these commands, and your admin password for others (administrative commands).

4.2 Client Commands

The commands in this section are meant to be run by someone acting in a client role. They require the client password, which is the same password that is required to submit a job. Client commands will also accept the administrator or cluster administrator passwords.

Listing Optimization Jobs

Optimization jobs running on a Compute Server cluster can be listed by using the `jobs` command. For example:

```
> grbcluster --server=server1 --password=pass jobs
```

JOBID	ADDRESS	STATUS	#Q	STIME	PRIQ
d2a6c505	server1	RUNNING		2017-10-04 20:43:21	0

Note that you can get more information by using the `--long` flag. With this flag, you will also display the complete job ID that is unique instead of the short ID.

```
> grbcluster --server=server1 --password=pass jobs --long
```

JOBID	ADDRESS	STATUS	#Q	STIME	USER	PRIQ	RUNTIME	PID	HOST	IP
d2a6c505-...	server1	RUNNING		2017-10-04 20:43:21	user1	0	8.0.0	4920	machine1	xxx...

The `jobs` command only shows jobs that are currently running. To obtain information on jobs that were processed recently, run the `recent` command:

```
> grbcluster --server=server1 --password=pass recent
```

JOBID	ADDRESS	STATUS	STIME	USER	OPT
64af5552	server1	COMPLETED	2017-10-06 17:58:30	user1	OPTIMAL

The information displayed by the `jobs` and `recent` commands can be changed using the `--view` flag. The default view for the two commands is the `status` view. Alternatives are:

- `status` - List all jobs and their statuses
- `model` - List all jobs, and include information about the models solved
- `simplex` - List jobs that used the SIMPLEX algorithm
- `barrier` - List jobs that used the BARRIER algorithm
- `mip` - list jobs that used the MIP algorithm

For example, the `model` view gives details about the model, including the number of rows, columns and nonzeros in the constraint matrix:

```
> grbcluster --server=server1 --password=pass recent --view=model
```

JOBID	STATUS	STIME	SOLVE	ROWS	COLS	NONZ	ALG	OBJ	DURATION
64af5552	COMPLETED	2017-10-06 17:58:30	COMPLETED	396	322	1815	MIP	1.2000126e+09	52.17s

To get an explanation of the meanings of the different fields within a view, add the `--describe` flag. For example:

```
> grbcluster recent --view=model --describe
```

JOBID	- Unique job ID, use --long to display full ID
STATUS	- Job status
STIME	- Job status updated time

```

SOLVE      - Solve status
ROWS       - Number of rows
COLS       - Number of columns
NONZ       - Number of non zero
ALG        - Algorithm MIP, SIMPLEX or BARRIER
OBJ        - Best objective
DURATION   - Solve duration

```

For a Mixed-Integer Program (MIP), the `mip` view provides progress information for the branch-and-cut tree. For example:

```

> grbcluster --server=server1 --password=pass recent --view=mip
JOBID  STATUS  STIME          OBJBST          OBJBND          NODCNT SOLCNT  CUTCNT  NODLFT
64af5552 COMPLETED 2017-10-06 17:58:30 1.2000126e+09 1.200000244173974e+09 178942 10      0      6046

```

Again, `--describe` explains the meanings of the different fields:

```

> grbcluster recent --view mip --describe
JOBID      - Unique job ID, use --long to display full ID
STATUS     - Job status
STIME      - Job status updated time
OBJBST     - Current best objective
OBJBND     - Current best objective bound
NODCNT     - Current explored node count
SOLCNT     - Current count of feasible solutions found
CUTCNT     - Current count of cutting planes applied
NODLFT     - Current unexplored node count
DURATION   - Solve duration

```

Note that the `jobs` command provides live status information, so you will for example see current MIP progress information while the solve is in progress.

The other views (`simplex` and `barrier`) are similar, although of course they provide slightly different information.

Accessing Job Logs

Gurobi Optimizer log output from a previous or currently running job can be retrieved by using the `log` command. For example:

```

> grbcluster --server=server1 --password=pass log 78400806

MIR: 3
Flow cover: 43

Explored 178942 nodes (1207578 simplex iterations) in 93.24 seconds
Thread count was 2 (of 2 available processors)

Solution count 10: 1.20001e+09 1.35001e+09 1.50001e+09 ... 1.65835e+09

Optimal solution found (tolerance 1.00e-04)
Best objective 1.200012600000e+09, best bound 1.200000244174e+09, gap 0.0010%

```

The argument to this command is the `JOBID` for the job of interest (which can be retrieved using the `jobs` command), you can use the full ID or the short ID. If you don't specify a `JOBID`, the command will display the log for the last job submitted.

The `log` command accepts the following arguments:

Usage:

```
grbcluster log <JOBID> [flags]
```

Important flags:

-b, --begin	Display log from the beginning
-f, --continuous	Display log continuously until job completion
-n, --lines int	Display only the last n lines (default 10)

For example, to get the entire log, from the beginning of the job, use the **-b** (or **--begin**) flag.

```
> grbcluster --server=server1 --password=pass log 78400806 -b
```

You can get a continuous feed of the log for a running optimization job with the **-f** (or **--continuous**) flag.

Accessing Job Parameters

The Gurobi Optimizer provides a number of parameters that can be modified by the user. The **params** command allows you to inspect the values of these parameters in a Compute Server job:

```
> grbcluster --server=server1 --password=pass params e7022667
```

```
TimeLimit= 60
```

The argument to this command is the **JOBID** for the job of interest (which can be retrieved using the **jobs** command), you can use the full ID or the short ID. If you don't specify a **JOBID**, the command will display the changed parameters of the last job submitted.

The following example illustrates how the **grbcluster params** command can be used in practice. The first step is to start an optimization job on a Compute Server cluster with one modified parameter:

```
> gurobi_cl --server=server1 --password=pass TimeLimit=120 glass4.mps
```

Once the job starts, you can use the **grbcluster jobs** command to retrieve the associated **JOBID** (or you can read it off from the output of **gurobi_cl**). For jobs that have been already processed, you would run the **recent** command instead.

```
> grbcluster --server=server1 --password=pass jobs
JOBID  ADDRESS STATUS #Q  STIME          PRIO
e88a496f server1 RUNNING    2017-10-06 23:09:34 0
```

Once you obtain the **JOBID**, the **params** command shows the modified parameter settings for the job:

```
> grbcluster --server=server1 --password=pass params e88a496f
```

```
TimeLimit= 120
```

The full list of Gurobi parameters can be found in the *Parameters* section of the [Gurobi Reference Manual](#).

Listing Cluster Nodes

The `nodes` command provides a list of nodes in the cluster, along with status information on those nodes. For example:

```
> grbcluster nodes --server=server1 --password=pass
```

ADDRESS	STATUS	TYPE	LICENSE	#Q	#R	JL	IDLE	%MEM	%CPU
server1	ALIVE	COMPUTE	VALID	0	0	2	23h13m	9.77	0.00
server2	ALIVE	COMPUTE	VALID	0	0	2	23h1m	8.75	0.00

The `--server` flag can point to any node in the cluster.

You can also get more information using the `--long` flag:

```
> grbcluster nodes --server=server1 --password=pass --long
```

ADDRESS	STATUS	TYPE	LICENSE	PROCESSING	#Q	#R	JL	IDLE	%MEM	%CPU	STARTED	RUNTIMES	VERSION
server1	ALIVE	COMPUTE	VALID	ACCEPTING	0	0	2	23h13m	9.77	0.00	2017-09-28 00:03:24	[8.0.0]	8.0.0
server2	ALIVE	COMPUTE	VALID	ACCEPTING	0	0	2	23h1m	8.75	0.00	2017-09-28 00:16:11	[8.0.0]	8.0.0

Add the `--describe` flag to see an explanation of each field:

```
> grbcluster nodes --describe
```

ADDRESS	- Node address
STATUS	- Node status (ALIVE, FAILED, JOINING, LEAVING, DEGRADED)
TYPE	- Node type (COMPUTE: compute server, WORKER: distributed worker)
GRP	- Group name for job affinity (not displayed if empty or restricted)
LICENSE	- License status (N/A, VALID, INVALID, EXPIRED)
PROCESSING	- Processing state (ACCEPTING, DRAINING, STOPPED), use --long
#Q	- Number of jobs in queue
#R	- Number of jobs running
JL	- Job Limit (maximum number of running jobs)
IDLE	- Idle time since the last job execution (in minutes)
%MEM	- Percentage of memory currently used on the machine
%CPU	- Percentage of CPU currently used on the machine
STARTED	- Node start time, use --long
RUNTIMES	- Deployed runtime versions, use --long
VERSION	- Remote Services Agent version, use --long

Troubleshooting Connectivity Issues

You can test to see if a Remote Services node is reachable with the `ping` command:

```
> grbcluster ping --server=server1 --password=pass
```

Node is not reachable

The `latency` command provides additional detail:

```
> grbcluster latency --server=server1 --password=pass
```

ADDRESS	LATENCY	NBERR
server1	1.12813ms	0
server2	1.218103ms	0

This will display the latency from the client machine to each node in the cluster.

Add the `--describe` flag to see an explanation of each field:

```
> grbcluster latency --describe
```

ADDRESS	- Node address
LATENCY	- latency between the local client and a node
NBERR	- Number of errors

4.3 Administrative Commands

The commands in this section are meant to be run by someone acting as an administrator. They require the administrator password, and they will also accept the cluster administrator password.

Listing Cluster Licenses

The `licenses` command displays license status information for each node in a cluster:

```
> grbcluster licenses --server=server1 --password=admin
```

ADDRESS	STATUS	TYPE	KEY	EXP	ORG	USER	APP	VER	CS	DL	ERROR
server1	VALID	NODE			gurobi			8	true	0	
server2	VALID	NODE			gurobi			8	true	0	

Add the `--describe` flag to see an explanation of each field:

```
> grbcluster licenses --describe
```

ADDRESS	- Node address
STATUS	- license status
TYPE	- License type
KEY	- License Cloud Key
EXP	- License expiration
ORG	- Assigned organization
USER	- Assigned username
APP	- Assigned application name
VER	- Maximum runtime version supported
CS	- Indicate if Compute Server features are enabled
DL	- Maximum number of workers for a distributed job (Distributed Limit)
ERROR	- License error message

If a node has an `INVALID` license, you can run the following command to learn more:

```
> grbcluster licenses --server=server1 --password=admin
```

ADDRESS	STATUS	TYPE	KEY	EXP	ORG	USER	APP	VER	CS	DL	ERROR
server1	INVALID	NODE							false	0	No Gurobi license found...

Note that the `licenses` command can be used at any time to check the validity and attributes of licenses on all the nodes of the cluster (expiration date, distributed limit etc).

You'll need to supply the administrator password to run this command.

Changing the Job Limit

Each node of a Remote Services has a job limit, which indicates the maximum of jobs that can be run simultaneously on that node. This job limit can be changed using the `grbcluster config` command, together with the `--job-limit=` flag. For example, to change the job limit to 5:

```
> grbcluster config --server=server1 --password=admin --job-limit=5
```

Changes to the job limit parameter only apply to the specified node; other nodes in the cluster are unaffected. Once changed, the new value will persist, even if you stop and restart the node.

Recall that you can run the `nodes` command to view the current job limits for each node in a cluster:

```
> grbcluster --server=server1 --password=admin nodes
ADDRESS STATUS TYPE    LICENSE #Q #R JL IDLE %MEM %CPU
server1 ALIVE  COMPUTE VALID    0  0  2  29m  9.97  0.00
server2 ALIVE  COMPUTE VALID    0  0  2  30m  8.86  1.00
```

The JL column shows the job limit, which is 2 for both nodes in the cluster in this example. We can change the limit for one node:

```
> grbcluster config --server=server1 --password=admin --job-limit=5
```

By rerunning the `nodes` command, we can see that the limit for `server1` has been changed to 5:

```
> grbcluster --server=server1 --password=admin nodes
ADDRESS STATUS TYPE    LICENSE #Q #R JL IDLE %MEM %CPU
server1 ALIVE  COMPUTE VALID    0  0  5  1h7m 10.12  0.00
server2 ALIVE  COMPUTE VALID    0  0  2  1h6m  8.92  0.00
```

Aborting Jobs

Jobs that are running on a Compute Server can be aborted by using the `abort` command. For example:

```
> grbcluster --server=server1 --password=admin abort e7022667
```

The following steps illustrate how you would start and subsequently abort a job. First, use the Gurobi command-line tool (`gurobi_cl`) to start a long-running optimization job on your Compute Server:

```
> gurobi_cl --server=server1 --password=pass glass4.mps
```

Once the job starts, you can use the `grbcluster jobs` command to retrieve the associated JOBID (or you can read it off from the output of `gurobi_cl`):

```
> grbcluster --server=server1 --password=pass jobs
JOBID ADDRESS STATUS #Q STIME Prio
8f9b15d9 server1 RUNNING 2017-10-10 17:30:33 0
```

The full or short JOBID can be used to abort the job as follows

```
> grbcluster --server=server1 --password=admin abort 8f9b15d9
```

If no JOBID is specified, the most recently started job will be aborted.

After the abort command is issued, the status of the job can be retrieved using the `recent` command:

```
> grbcluster --server=server1 --password=pass recent
JOBID ADDRESS STATUS STIME USER OPT
8f9b15d9 server1 ABORTED 2017-10-10 17:41:33 user1 OPTIMAL
```

As you can see, the status of the job has changed to `ABORTED`.

Programming with Remote Services

While applications that use Remote Services can generally be built without having to consider where they will be run, there are a few aspects of Remote Services that programmers should be aware. These are covered in this section.

5.1 Using an API to Create a Compute Server Job

As was noted earlier, a Remote Services client program will always need to be told how to reach the Remote Services cluster. This can be done in two ways. The first is through a license file. This approach is described in an [earlier discussion](#). It requires no changes to the application program itself. The same program can perform optimization locally or remotely, depending on the settings in the license file.

Your second option for specifying the desired Compute Servers is through API calls. You would first construct an **empty environment** (using `GRBEmptyenv` in C or the appropriate `GRBEnv` constructor in the object-oriented interfaces), then set the appropriate parameters on this environment (typically `ComputeServer` and `ServerPassword`), and then start the empty environment (using `GRBstartenv` in C or `env.start()` in the object-oriented interfaces).

To give a simple example, if you'd like your Python program to offload the optimization computation to a Compute Server named `server1`, you could say:

```
env = Env(empty=True)
env.setParam(GRB.Param.ComputeServer, "server1:61000")
env.setParam(GRB.Param.ServerPassword, "passwd")
env.start()
model = read("misc07.mps", env)
model.optimize()
```

An equivalent Java program would look like this:

```
GRBEnv env = new GRBEnv(true);
env.set(GRB.StringParam.ComputeServer, "server1:61000");
env.set(GRB.StringParam.ServerPassword, "passwd");
GRBModel model = new GRBModel(env, "misc07.mps");
model.optimize();
```

We refer you to the [Gurobi Reference Manual](#) for details on these routines.

5.2 Performance Considerations on a Wide-Area Network (WAN)

While using Gurobi Compute Server doesn't typically require you to make any modifications to your code, performance considerations can sometimes force you to do some tuning when your client and server are connected by a slow network (e.g., the internet). We'll briefly talk about the source of the issue, and the changes required to work around it.

In a Gurobi Compute Server, a call to a Gurobi routine often results in a network message between the client and the server. While each individual message is not that expensive, sending hundreds or thousands of messages can be quite time-consuming. Compute Server makes heavy use of caching to reduce the number of such messages, and this caching generally works well, so you don't need to be too concerned about it.

Furthermore, when building a model, our *lazy update* approach avoids the issue entirely. You should feel free to build your model one constraint at a time, for example. Your changes are communicated to the server in one large message when you request a model update.

Having said that, we should add that not all methods are batched or cached. As a result, we suggest that you avoid doing the following things:

- Retrieving the non-zero values for individual rows and columns of the constraint matrix (using, for example, `GRBgetconstrs` in C, `GRBModel::getRow` in C++, `GBModel.getRow` in Java, `GRBModel.GetRow` in .NET, and `Model.getRow` in Python).
- Retrieving individual string-valued attributes.

Of course, network overhead depends on both the number of messages that are sent and the sizes of these messages. We automatically perform data compression to reduce the time spent transferring very large messages. However, as you may expect, you will notice some lag when solving very large models over slow networks.

5.3 Callbacks

As you might imagine, since the actual optimization task runs on a remote system in a Compute Server environment, Gurobi callbacks give different behavior than they do when the task runs locally. In particular, callbacks are both less frequent and more restrictive. You will only receive `MESSAGE`, `BARRIER`, `SIMPLEX`, `MIP`, `MIPSOL` and `MULTIOBJ` callbacks; you will not receive `PRESOLVE` or `MIPNODE` callbacks. As a result, you will only have access to a subset of the callback information that you would be able to obtain when running locally. You can still request that the optimization be terminated from any of the callbacks you receive. Please refer to the `Callback Code` section of the [Gurobi Reference Manual](#) for more information on the various callback codes.

5.4 Developing for Compute Server

With only a few exceptions, using Gurobi Compute Server requires no changes to your program. This section covers the exceptions. We'll talk about program robustness issues that may arise specifically in a Compute Server environment, and we'll give a full list of the Gurobi features that aren't supported in Compute Server.

Coding for Robustness

Client-server computing introduces a few robustness situations that you wouldn't face when all of your computation happens on a single machine. Specifically, by passing data between a client and a server, your program is dependent on both machines being available, and on an uninterrupted network connection between the two systems. The queuing and load balancing capabilities of Gurobi Compute Server can handle the vast majority of issues that may come up, but you can take a few additional steps in your program if you want to achieve the maximum possible robustness.

The one scenario you may need to guard against is the situation where you lose the connection to the server while the portion of your program that builds and solves an optimization model is running. Gurobi Compute Server will automatically route queued jobs to another server, but jobs that are running when the server goes down are interrupted (the client will receive a `NETWORK` error). If you want your program to be able to survive such failures, you will need to architect it in such a way that it will rebuild and resolve the optimization model in response to a `NETWORK` error. The exact steps for doing so are application dependent, but they generally involve encapsulating the code between the initial Gurobi environment creation and the last Gurobi call into a function that can be reinvoked in case of an error.

Features Not Supported in Compute Server

As noted earlier, there are a few Gurobi features that are not supported in Compute Server. We've mentioned some of them already, but we'll give the full list here for completeness. You will need to avoid using these features if you want your application to work in a Compute Server environment.

The unsupported features are:

- **Lazy constraints:** While we do provide MIPSOL callbacks, we don't allow you to add lazy constraints to cut off the associated MIP solutions.
- **User cuts:** The MIPNODE callback isn't supported, so you won't have the opportunity to add your own cuts. User cuts aren't necessary for correctness, but applications that heavily rely on them may experience performance issues.
- **Multi-threading within a single Gurobi environment:** This isn't actually supported in Gurobi programs in general, but the results in a Compute Server environment are sufficiently difficult to track down that we wanted to mention it again here. All models built from an environment share a single connection to the Compute Server. This one connection can't handle multiple simultaneous messages. If you wish to call Gurobi from multiple threads in the same program, you should make sure that each thread works within its own Gurobi environment.
- **Advanced simplex basis routines:** The C routines that work with the simplex basis (`GRBFSolve`, `GRBBSolve`, `GRBBinvColj`, `GRBBinvRowi`, and `GRBgetBasisHead`) are not supported.

5.5 Distributed Algorithms

Gurobi Remote Services allow you to perform distributed optimization. All you need is a cluster with more than one node. The nodes can be either Compute Server or distributed worker nodes. Ideally these nodes should all give very similar performance. Identical performance is best, especially for distributed tuning, but small variations in performance won't hurt overall results too much.

Choosing an Appropriate Cluster

Before launching a distributed optimization job, you should run the `grbcluster nodes` command to make sure the cluster contains more than one live machine:

```
> grbcluster nodes --server=server1:port --password=pass
```

If you see multiple live nodes, then that cluster is good to go:

ADDRESS	STATUS	TYPE	LICENSE	#Q	#R	JL	IDLE	%MEM	%CPU
server1:61000	ALIVE	COMPUTE	VALID	0	0	1	43m0s	42.67	2.53
server2:61000	ALIVE	WORKER	N/A	0	0	1	<1s	42.67	1.76

We should reiterate a point that was raised earlier: you do not need a Gurobi license to run Gurobi Remote Services on a machine. While some services are only available with a license, any machine that is running Gurobi Remote Services will provide the Distributed Worker service.

Running A Distributed Algorithm

Running a distributed algorithm is simply a matter of setting the appropriate Gurobi parameter. Gurobi supports distributed MIP, concurrent LP and MIP, and distributed tuning. These are controlled with three parameters: `DistributedMIPJobs`, `ConcurrentJobs`, and `TuneJobs`, respectively. These parameters indicate how many distinct distributed worker jobs you would like to start. Keep in mind that the initial Compute Server job will act as the first worker.

To give an example, if you have a cluster consisting of two machines (`server1` and `server2`), and if you set `TuneJobs` to 2 in `grbtune`...

```
> grbtune --server=server1:61000 --password=passwd TuneJobs=2 misc07.mps
```

...you should see output that looks like the following...

```
Capacity available on 'server1:61000' - connecting...
```

```
...
```

```
Using Compute Server as first worker
```

```
Started distributed worker on server2:61000
```

```
Distributed tuning: launched 2 distributed worker jobs
```

This output indicates that two worker jobs have been launched, one on machine `server1` and the other on machine `server2`. These two jobs will continue to run until your tuning run completes. Similarly, if you launch distributed MIP...

```
> gurobi_cl --server=server1:61000 --password=passwd DistributedMIPJobs=2 misc07.mps
```

...you should see the following output in the log...

```
Using Compute Server as first worker
```

```
Started distributed worker on server2:61000
```

```
Distributed MIP job count: 2
```

Note that distributed workers are allocated on a first-come, first-served basis, so if multiple users are sharing a cluster, you should be prepared for the possibility that some or all of your distributed workers may be busy when you request them. Your program will grab as many as it can, up to the requested count. If none are available, it will return an error.

Using a Separate Manager

While distributed workers always need to be part of a Remote Services cluster, note that the manager itself does not. Any machine that is licensed to run distributed algorithms can act as the manager. You simply need to set `WorkerPool` and `WorkerPassword` parameters to point to the Remote Services cluster that contains your distributed workers. To give an example:

```
> gurobi_cl WorkerPool=server1:61000 WorkerPassword=passwd DistributedMIPJobs=2 misc07.mps
```

...you should see the following output in the log...

```
Started distributed worker on server1:61000
Started distributed worker on server2:61000
```

```
Distributed MIP job count: 2
```

In this case, the distributed computation is managed by the machine where you launched this command, and the two distributed workers come from your Remote Services cluster.

Compute Server Considerations

As noted earlier, [Gurobi Compute Servers](#), can be used for distributed optimization as well. Compute Servers offer a lot more flexibility than distributed workers, though, so they require a bit of additional explanation.

The first point you should be aware of is that one Compute Server node can actually host multiple distributed worker jobs. Compute Server nodes allow you to set a limit on the number of jobs that can run simultaneously. Each of those jobs can be a distributed worker. For example, if you have a cluster that contains a pair of Compute Server nodes, each with a job limit of 2, then issuing the command...

```
> gurobi_cl --server=server1:61000 --password=passwd DistributedMIPJobs=3 misc07.mps
```

...would produce the following output...

```
Capacity available on 'server1:61000' - connecting...
...
Using Compute Server as first worker
Started distributed worker on server2:61000
Started distributed worker on server1:61000
```

Compute Server assigns a new job to the machine with the most available capacity, so assuming that the two servers are otherwise idle, the first distributed worker job would be assigned to `server1`, the second to `server2`, and the third to `server1`.

5.6 Distributed Algorithm Considerations

So far in this section, we've focused almost entirely on configuration and setup issues for the distributed algorithms in this section. These algorithms have been designed to be nearly indistinguishable from the single machine versions. Our hope is that, if you know how to use the single machine version, you'll find it straightforward to use the distributed version. The distributed algorithms respect all of the usual parameters. For distributed MIP, you can adjust strategies, adjust tolerances, set limits, etc. For concurrent MIP, you can allow Gurobi to choose the settings for each machine automatically or you can use `concurrent environments` to make your own choices. For distributed tuning, you can use the usual tuning parameters, including `TuneTimeLimit`, `TuneTrials`, and `TuneOutput`.

Performance Across Distributed Workers

There are a few things to be aware of when using distributed algorithms, though. One relates to relative machine performance. As we noted earlier, distributed algorithms work best if all of the workers give very similar performance. For example, if one machine in your worker pool were much slower than the others in a distributed tuning run, any parameter sets tested on the slower machine would appear to be less effective than if they were run on a faster machine. Similar considerations apply for distributed MIP and distributed concurrent. We strongly recommend that you use machines with very similar performance. Note that if your machines have similarly performing cores but different numbers of cores, we suggest that you use the `Threads` parameter to make sure that all machines use the same number of cores.

Callbacks

Another difference between the distributed algorithms and our single-machine algorithms is in the callbacks. The distributed MIP and distributed concurrent solvers do not provide the full range of callbacks that are available with our standard solvers. They will only provide the `MIP`, `MIPNODE`, and `POLLING` callbacks. See the `Callback` section of the [Gurobi Reference Manual](#) for details on the different callback types.

Logging

The distributed algorithms provide slightly different logging information from the standard algorithms. Consult the `Distributed MIP Logging` section of the [Gurobi Reference Manual](#) for details.

5.7 Cluster REST API

Each node in the cluster also exposes a REST API in order to support advanced integration. The API follows standard REST principles and can be used in various languages and tools (Java, Python, Node, curl...).

The base URL of the API is the node address followed by the API prefix `/api/v1`. For example, if a node is running on `server1` using HTTP on the default port, the base URL will be:

```
http://server1/api/v1
```

If it is using HTTPS on a custom port 61000, the base URL will be

```
https://server1:61000/api/v1
```

The API is composed of several endpoints. In order to access an API endpoint, you will also need to provide a the password in the header `X-GUROBI-CSPASSWORD`. The password can be the client or administrator password depending on the endpoint.

We distinguish between the cluster and the node endpoints. Cluster endpoints provide cluster-wide APIs and any of the nodes from the same cluster can be used. On the other hand, node endpoints provide node specific APIs, for example to manage the configuration of a node or access a running or recently completed job on a specific node. Here is a summary of the cluster endpoints:

GET `/cluster/licenses`: Lists the licenses

GET `/cluster/nodes`: Lists the nodes

GET `/cluster/jobs`: Lists the jobs

GET `/cluster/jobs/id`: Returns a job description

DELETE `/cluster/jobs/id/processing`: Aborts a job - Administrator password is required.

Here is a summary of the node endpoints:

GET `/ping`: Pings a node

GET `/config`: Gets current configuration

POST `/config`: Updates the configuration - Administrator password is required.

GET `/jobs/id/log`: Returns the log of an active job

GET `/jobs/id/metrics`: Returns the metrics of an active job

GET `/jobs/id/parameters`: Returns the parameters of an active job

The detailed and interactive documentation is also provided using the Swagger format and available directly on a node, for example:

```
http://server1/swagger.html
```

Using Remote Services with Gurobi Instant Cloud

Our Gurobi Instant Cloud product is built on top of Amazon's Elastic Compute Cloud (EC2) platform. When you launch an Instant Cloud instance, we launch a machine on EC2 and start Gurobi Remote Services for you on that machine. You also have the option of launching multiple machines, in which case we'll create a Remote Services cluster for you. Once you have [set up your client](#) with a client license file, you will be able to use `grbcluster` to [monitor](#) and [administer](#) the cluster. Note, however, that cluster administrative commands are not accessible, since the Gurobi Instant Cloud Manager already plays the cluster administrator role. Note also that communication with your Instant Cloud instance will always use HTTPS, and it will go through a [region router](#).

6.1 Client Setup

To access the cluster started by Instant Cloud, you first need to download the machine or pool license file from the Instant Cloud manager. You can download the default license file from the license panel, the pool license from the pool panel, or the machine license from the machine panel. Then, you need to save this file in your home directory or in one of the following locations:

- `C:\gurobi\` on Windows
- `/opt/gurobi/` on Linux
- `/Library/gurobi/` on Mac OS

You can also set the environment variable `GRB_LICENSE_FILE` to point to this file.

6.2 Client Commands

Once you have set up your client license file and started an Instant Cloud instance, you can use **grbcluster** to list the nodes in your cluster or issue other client commands. Instances can be started by submitting a job through the **gurobi_cl** command-line tool, through the Gurobi programming language APIs, or manually through the Instant Cloud Manager website.

If you try to run **grbcluster** without first starting an instance, you will get the following error:

```
fatal : Instant Cloud pool default has no machines
```

If your instance is in the process of starting, you will get the following error:

```
fatal : Instant Cloud pool default is not ready
```

If your instance is up and running, **grbcluster** will list the nodes in your cluster:

```
> grbcluster nodes
ADDRESS      STATUS TYPE   GRP                LICENSE #Q #R JL IDLE  %MEM  %CPU
ip-172-31-31-180 ALIVE  COMPUTE m-HkQmbubhWH1g7m VALID   0  0  2  12m0s 27.08 1.98
ip-172-31-62-109 ALIVE  COMPUTE m-HJSXmb_-2WBkLX VALID   0  0  2  12m0s 27.49 0.00
```

To obtain additional details (about the license file, the cloud pool, or the name of the server), you can use the verbose mode with the **-v** flag:

```
> grbcluster -v nodes
verb : Reading license file /licenses/gurobi.lic
verb : Accessing Instant Cloud pool 999999-pool5
verb : Using remote services on node ip-172-31-31-180
ADDRESS      STATUS TYPE   GRP                LICENSE #Q #R JL IDLE  %MEM  %CPU
ip-172-31-31-180 ALIVE  COMPUTE m-HkQmbubhWH1g7m VALID   0  0  2  12m0s 27.08 1.98
ip-172-31-62-109 ALIVE  COMPUTE m-HJSXmb_-2WBkLX VALID   0  0  2  12m0s 27.49 0.00
```

You can use **grbcluster** to perform all of the same [client commands](#) on an Instant Cloud cluster that you'd perform on a cluster running locally. You can monitor running and recently processed jobs, access log files, view parameters, etc.

6.3 Administrative Commands

Gurobi Instant Cloud allows you to perform administrative commands (to abort a job, change the job limit, etc.), but you'll need to retrieve the administrator password to do so. The administrator password can be retrieved from the [Instant Cloud Manager](#). For a running machine, you will find the administrator password in the **Machines** area, within machine details under the **PASSWORDS** tab. If you have no running machine, you can find it in the **Settings** area, under passwords settings.

For example, you can abort a specific job:

```
> grbcluster --password=XXXXXXXX abort 3545d9de-8de4-4666-9491-5d60e2e56186
```

Or, you can set the job limit of a specific Compute Server node:

```
> grbcluster --server=ip-172-31-62-109 --password=XXXXXXXX config --job-limit=10
```

6.4 Region Router

Starting with version 8.0, all communications between clients and the Gurobi Instant Cloud use the HTTPS protocol. This means that your communications are secured and encrypted using standard internet protocols. In addition, Gurobi servers enforce the latest encryption policies (TLS v1.2 and above only). For better security, the dedicated machines started by Instant Cloud on your behalf cannot be accessed directly. All communications must transit through a secured and highly-available region router acting as a reverse proxy. This also facilitates the integration with clients, as only the standard HTTPS protocol and standard port 443 need to be open if a firewall is in place.

The region router is automatically detected and used based on the pool definition or the machine license file.

Migrating from Previous Releases

While the new Compute Server provides a similar set of features to the previous release, a few of the mechanics are quite different.

7.1 Referring to Compute Servers

One important difference between version 8.0 and previous versions relates to clustering. Previous releases supported queueing and load balancing among multiple servers, but this clustering was handled on the client side. Your client program would explicitly list the Compute Server nodes that your job could run on, and the client would negotiate with those servers to determine where and when it ran. In the new Compute Server, you [form explicit clusters of nodes](#), and your client simply needs to refer to any member of the cluster in order to run a job on any node in the cluster.

Another important difference is in how you refer to Compute Server nodes. While it has always been the case that you need to provide both the name of the Compute Server node and the port number that Remote Services is listening on, those were separated in previous releases. For example, you might say:

```
> gurobi_cl --server=server1 --port=61000
```

In version 8.0, communication happens through a REST API, so you now use standard HTTP syntax to refer to servers:

```
> gurobi_cl --server=server1:61000
```

...OR...

```
> gurobi_cl --server=http://server1:61000
```


7.2 Installation - Ports and Firewalls

In previous releases, `grb_rs` required a large range of ports to be open (61000 through 65000 by default). Version 8.0 only requires a single port, which simplifies the installation and the setup of firewalls. The default port depends on the protocol: port 80 for HTTP and port 443 for HTTPS. In order to use these default ports, you will need to be an administrator to start `grb_rs`, and also make sure that they are not already in use. You can also choose a custom port (e.g., port 61000).

Refer to the sections about starting `grb_rs` as a [standard process](#) or [service](#) for more details.

7.3 Installation - Encryption

In previous releases, data exchanged between a client and `grb_rs` was encrypted by default with a proprietary protocol using AES encryption. Version 8.0 supports standard TLS encryption over HTTPS. However, you will need to explicitly [activate](#) this protocol and setup certificates. `grb_rs` can also help you by generating [self-signed certificates](#), if necessary.

7.4 Command-line options in gurobi_cl

Several options of `gurobi_cl` have been removed and replaced as the following:

- `--status`: Listing the jobs and the status of a node is now available using the `grbcluster` command line tool. The [command](#) `grbcluster nodes` lists all the nodes of the cluster with some metrics (CPU and memory usage), number of jobs in queue and running, processing status, license status etc. The [command](#) `grbcluster jobs` lists all the jobs running in the cluster. Note that you can also list the license status with the [command](#) `grbcluster licenses`.
- `--killjob`: Killing a job is supported by the `grbcluster abort` [command](#).
- `--joblimit`: Changing the job limit of each node is now supported by the `grbcluster config` [command](#).
- `--newadminpassword`: It is not possible to change dynamically the password anymore. If you need to change a password, you will need to stop the node, edit the `grb_rs.cnf` configuration file, and finally restart the node. Also, all the nodes in the cluster should have the same passwords to avoid any inconsistent behavior.

7.5 Distributed Optimization

If you used distributed algorithms in previous releases, you'll find that the meaning of the `WorkerPool` parameter has changed. In previous releases, the client program was responsible for listing all machines that could be used as distributed workers. In the new release, the workers should all belong to a single Remote Services [cluster](#), and your client program simply needs to point to any node of the cluster.

Usage:

grb_rs [flags]	Start the remote services as a standard process
grb_rs --help	Display usage
grb_rs command [flags]	Execute a specific command
grb_rs command --help	Display more information about a command

Flags can be set using --flag=value or the short form -f=value if defined.
A boolean flag can be enabled using --flag or the short form -f if defined.

Configuration Helper Commands:

hash	Hash a password
init	Clone the default data directory and configuration to current directory
token	Generate a cluster token
properties	Display help about configuration properties

Service Commands:

install	Install the service
restart	Start or restart the service (install the service if necessary)
start	Start the service (install the service if necessary)
stop	Stop the service
uninstall	Uninstall the service

With no command, grb_rs will start the remote services as a standard process and the following flags are available for quick configuration. The full list of properties can be displayed with the 'grb_rs properties' command and the properties can be set in the grb_rs.cnf configuration file.

Logging Flags:

--console-ts	Add timestamps to console log messages
--logfile string	Log to a rotating log file
--logfile-max-age int	Limit the rotating log file to a number of days
--logfile-max-size int	Limit the size of each file to a size in Mb
--no-console	Disable log to console
--syslog	Log to syslog or Windows event log
-v, --verbose	Enable verbose logging

Configuration Flags:

-c, --config string	Location of the configuration file (default: 'grb_rs.cnf')
--data string	Location of the data root directory (default: 'data')
--hostname	Overrides the public name on this name
--idle-shutdown int	Shutdown if the server is idle for more than the specified time limit (minutes)
--join string	Join a cluster using the specified cluster representative node address
--port int	Start the node on the given port
--service	Indicates if it is started by a service manager

<code>--worker</code>	Declare this node as a distributed worker
Security Flags:	
<code>--tls</code>	Use TLS for communication encryption
<code>--tlscert string</code>	Path to TLS certificate file
<code>--tlskey string</code>	Path to TLS key file
<code>--tls-insecure</code>	Use TLS but disable verification of certificates works with self-signed certificates
General Flags:	
<code>--version</code>	Display version information
<code>--help</code>	Display usage

Appendix B: grb_rs - Configuration Properties

The following list of properties can also be displayed using the `grb_rs properties` command.

ADMINPASSWORD: Type string. Client password to administrate the jobs. The password can be in clear or can be hashed using `'grb_rs hash'` command for better security.

AWS: Type bool, use `--aws` to override on the command line. Enable AWS configuration using `ec2 user-data`.

AWS_HOSTNAME_MODE: Type string. Indicates how to get the node name on AWS: 'public' or 'private'. The public mode will assign the EC2 public DNS name, whereas the private mode will assign the base name of the private DNS name. The private mode is used with a Gurobi router.

CLIENT_DETAILS_ADMIN: Type bool. Indicates client details such as host, IP are only accessible as an admin user. When a job is submitted, the client hostname, IP, and process ID are recorded. By default, this information is displayed to any user running the command line tool `grbcluster` or the REST API. If this property is set to true, only the administrator will be able to access this information.

CLOUDKEY: Type string. Cloud license key.

CLUSTER_ADMINPASSWORD: Type string. Client password to administrate the cluster. The password can be in clear or can be hashed using `'grb_rs hash'` command for better security.

CLUSTER_TOKEN: Type string. Unique cluster identifier. The token is an encrypted key to let nodes communicate between each other. All nodes of a cluster must have the same token. Use `'grb_rs token'` command to generate a new token.

CONSOLE_TS: Type bool, use `--console-ts` to override on the command line. Add timestamps to console log messages.

DATA_DIR: Type string (default data), use `--data` to override on the command line. Root directory to store remote services data.

DEGRADED_TIMEOUT: Type int (default 60). Timeout to evict a node that is DEGRADED from the cluster. 0 for no timeout.

FILE_DESCRIPTOR_LIMIT: Type int (default 2048). Maximum number of file descriptors.

FIXED_JOBLIMIT: Type bool. Indicates if the job limit can be changed once the node started.

GROUP: Type string. Node grouping for job affinity assignment.

HARDJOBLIMIT: Type int (default 0). A hard limit on the number of simultaneous client jobs. Certain jobs (those with priority 100) are allowed to ignore the JOBLIMIT, but they aren't allowed to ignore this limit. Client requests beyond this limit are queued. Use 0 to disable.

HOSTNAME: Type string, use `--hostname` to override on the command line. Advertised hostname of the cluster node.

IDLESHUTDOWN: Type int (default -1), use `--idle-shutdown` to override on the command line. Idle time limit (minutes) to trigger a shutdown of the server, -1 to disable.

IDLESHUTDOWN_COMMAND: Type string. Command to execute when the idle shutdown is reached, for example to shutdown the machine.

IDLESHUTDOWN_STOPPED: Type int (default -1). Idle time limit (minutes) to trigger a shutdown of the machine once the processing state is STOPPED, -1 to disable.

IDLETIMEOUT: Type int (default 0). Default idle timeout in seconds. If a job does not send a command for more than the timeout, it will be terminated. Use 0 to disable.

IGNOREPRIORITIES: Type bool. Disable job priority handling.

JOBLIMIT: Type int (default 2). A limit on the number of client jobs that are allowed to run on the server at a time. Client requests beyond this limit are queued.

JOIN: Type string, use `--join` to override on the command line. List of other nodes to join.

JOIN_TIMEOUT: Type int (default 20). Timeout for a successful join, use 0 to disable.

KEEPALIVE_TIMEOUT: Type int (default 60). Default keep alive timeout in seconds. If a job does not send a keep alive message for more than the timeout, it will be terminated.

LICENSEID: Type string. Cloud license ID.

LOGFILE: Type string, use `--logfile` to override on the command line. Enable logging to a rotating log file.

LOGFILE_MAX_AGE: Type int (default 5), use `--logfile-max-age` to override on the command line. Limit the rotating log file to a number of days.

LOGFILE_MAX_SIZE: Type int (default 500), use `--logfile-max-size` to override on the command line. Limit the size of each file to a size in Mb.

MAX_QUEUE: Type int (default 1000). Maximum number of jobs in the queue.

MAX_RECENT: Type int (default 50). Maximum number of executed jobs in the recent history.

NOQUEUE: Type bool. Disable job queueing.

NO_CONSOLE: Type bool, use `--no-console` to override on the command line. Disable the console log.

PASSWORD: Type string. Client password to access the cluster. The password can be in clear or can be hashed using 'grb_rs hash' command for better security.

PORT: Type int, use `--port` to override on the command line. Port number for the REST API.

REGISTRATION_PORT: Type int. Port used to register worker, 0 means a dynamic port.

SYSLOG: Type bool, use `--syslog` to override on the command line. Log to syslog or windows event log.

THREADLIMIT: Type int (default -1). Maximum number of threads used by a worker.

TLS: Type bool, use `--tls` to override on the command line. Enable TLS encryption protocol.

TLS_CERT: Type string, use `--tlscert` to override on the command line. Path to TLS certificate file. If not specified, a self-signed certificate will be generated.

TLS_INSECURE: Type bool, use `--tls-insecure` to override on the command line. Enable TLS encryption protocol but skip certificate verification. This mode can be used with self-signed certificate so that data is encrypted.

TLS_KEY: Type string, use `--tlskey` to override on the command line. Path to TLS key file. If not specified, a key will be generated to self-sign a certificate.

USERNAME_ADMIN: Type bool. Indicates that job username is only accessible as an admin user. When a job is submitted, the client process username is recorded. By default, this information is displayed to any user running the command line tool `grbcluster` or the REST API. If this property is set to true, only the administrator will be able to access this information.

VERBOSE: Type bool, use `--verbose` to override on the command line. Enable verbose logging.

WORKER: Type bool, use `--worker` to override on the command line. Declare the node as a distributed worker.

Appendix C: grbcluster

Usage:

```
grbcluster --help          Display usage
grbcluster command [flags] Execute a specific command
grbcluster command --help  Display more information about a command
```

Flags can be set using `--flag=value` or the short form `-f=value` if defined.
A boolean flag can be enabled using `--flag` or the short form `-f` if defined.

Client Commands:

```
jobs          List the active jobs
latency       Test node communication latency
log           Displays the log of a job
nodes         List the cluster nodes
params        List the changed parameters of a job
ping          Indicates if a node is reachable
recent        List the recently executed jobs
```

Administrator Commands:

```
abort         Abort a submitted job
config        Change the job limit property of a node
licenses      List the cluster licenses
```

Cluster Administrator Commands:

```
deploy        Deploy a Gurobi Optimizer runtime on one or several nodes
join          Request a node to join another node to form a cluster
leave         Request a node to leave the cluster
start         Enable job processing on a node
stop          Disable job processing on a node
undeploy      Undeploy a Gurobi Optimizer runtime from one or several nodes
```

Global Flags:

```
--console-ts  Add timestamps to console log messages
--help        Display usage
-v, --verbose  Enable verbose logging
--version     Display version information
```

If a valid Gurobi license file is accessible at the predefined locations or using the variable `GRB_LICENSE_FILE`, the license file will provide default values for some connection parameters (server, password, router). If the license file references a Gurobi Instant Cloud pool, it will resolve the connection parameters of the pool. However, the password of the administrator or the cluster administrator must be passed explicitly.

`grbcluster` is compatible with standard proxy settings using environment variables `HTTP_PROXY` and `HTTPS_PROXY`. `HTTPS_PROXY` takes precedence over `HTTP_PROXY` for https requests. The values may be either a complete URL or a "host[:port]", in which case the "http" scheme is assumed.

Usage:

<code>gurobi_cl --help</code>	Display usage
<code>gurobi_cl [flags] [param=value]* filename</code>	Optimize a model file
<code>gurobi_cl [flags]</code>	Execute a command

Gurobi parameters are documented in the Gurobi Reference Manual.

Flags can be set using `--flag=value` or the short form `-f=value` if defined.
A boolean flag can be enabled using `--flag` or the short form `-f` if defined.

Flags:

<code>-h, --help</code>	Display usage
<code>--license</code>	Display license information
<code>-t, --tokens</code>	List license tokens currently in use
<code>-v, --version</code>	Display version information

Compute Server Flags:

<code>--group string</code>	Cluster group name, overrides license file GROUP
<code>-p, --password string</code>	Password, overrides license file PASSWORD (default "pass")
<code>--priority int</code>	Job priority, overrides license file PRIORITY (default 0, min -100, max 100)
<code>-r, --router string</code>	Router URL, overrides license file property ROUTER
<code>-s, --server string</code>	Cluster representative node address, overrides license file COMPUTESERVER
<code>--tls-insecure</code>	Skip TLS certificate verification, overrides GRB_TLS_INSECURE variable

Instant Cloud Flags:

<code>--accessid string</code>	Access ID, overrides license file CLOUDACCESSID
<code>--secretkey string</code>	Secret Key, overrides license file CLOUDKEY
<code>--pool string</code>	Pool name, overrides license file POOL

If a valid Gurobi license file is accessible at the predefined locations or using the variable `GRB_LICENSE_FILE`, the license file will provide default values for some connection parameters (server, password, router). If the license file references a Gurobi Instant Cloud pool, it will resolve the connection parameters of the pool.

The server URL can also specify the protocol and the port:

<code>server.company.com</code>	Use HTTP on standard port 80
<code>server.company.com:61000</code>	Use HTTP on port 61000
<code>https://server.company.com</code>	Use HTTPS on standard port 443
<code>https://server.company.com:61000</code>	Use HTTPS on port 61000

If you wish to get the status of your compute server cluster, list the nodes and the jobs, or check the status of your licenses, please use the `grbcluster` command line tool. To learn more about `grbcluster`, type the following command:
`grbcluster --help`

gurobi_cl is compatible with standard proxy settings using environment variables HTTP_PROXY and HTTPS_PROXY. HTTPS_PROXY takes precedence over HTTP_PROXY for https requests. The values may be either a complete URL or a "host[:port]", in which case the "http" scheme is assumed.

Examples:

```
gurobi_cl misc07.mps
gurobi_cl Record=1 Method=2 ResultFile=p0033.sol InputFile=p0033.mst \
    InputFile=p0033.hnt.gz LogFile=p0033.log p0033.mps
gurobi_cl --server=server.company.com --password=pass misc07.mps
```

Visit www.gurobi.com/documentation/8.0 for further details on how to use this program.

Appendix E: Acknowledgement of 3rd Party Icons

The icons used in this document come from the [Open Security Architecture](#).

Appendix F: Open Source Component Licenses

Copyright (c) 2015, Dave Cheney <dave@cheney.net>
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The MIT License (MIT)

Copyright (c) 2014 Nate Finch

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to

communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed

as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity,

or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The MIT License (MIT)

Copyright (c) 2013 Ben Johnson

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN

CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2013 Julien Schmidt. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JULIEN SCHMIDT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2012 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (C) 2013-2016 by Maxim Bubliss <b@codemonkey.ru>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

gopsutil is distributed under BSD license reproduced below.

Copyright (c) 2014, WAKAYAMA Shirou
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the gopsutil authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

internal/common/binary.go in the gopsutil is copied and modified from golang/encoding/binary.go.

Copyright (c) 2009 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2012 Alex Ogier. All rights reserved.
Copyright (c) 2012 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The MIT License (MIT)

Copyright (c) 2014 Jeremy Saenz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2009 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright since 2015 Showmax s.r.o.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Copyright 2016 SmartBear Software

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at [apache.org/licenses/LICENSE-2.0] (<http://www.apache.org/licenses/LICENSE-2.0>)

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.