Server Specification of the Virtual Soccer Competition

This is a document describing the server infrastructure of the virtual RoboCup Humanoid League soccer competition. In case you feel that your team requires elements to run the robot controller which are not described in or in conflict with this document, please contact the TC as soon as possible and we will examine your request.

Hosting & Computing

The competition will be hosted using Amazon Web Services. Two types of virtual machines will be used for the competition:

- **Simulator Instance**: AWS g4dn.4xlarge instance
  - **GPU**: NVIDIA T4
  - **CPU**: 2nd Generation Intel Xeon Scalable (Cascade Lake) processors, 8 vCPUs (the instance type has 16 by default but is launched with one thread per core)
  - **Memory**: 64 GB
  - **Operating System**: Ubuntu Server 18.04 LTS, 64-bit x86
- **Robot Instance**: AWS g4dn.xlarge instance
  - **GPU**: NVIDIA T4
  - **CPU**: 2nd Generation Intel Xeon Scalable (Cascade Lake) processors, 4 vCPUs
  - **Memory**: 16 GB
  - **Operating System**: Ubuntu Server 18.04 LTS, 64-bit x86

Each robot has its own, dedicated robot instance ("robot VM").

Inside each robot VM a Docker image with the robot control software for exactly one robot will be running. There will be other processes running on the robot VM besides the ones in the Docker container but the footprint of these processes should be negligible.

If you would like to try your robot controller on AWS, you can use the public image of the Webots simulator provided by the Technical Committee. You need to select Ohio (us-east-2) as your region on AWS and then search for the latest AMI with `robocup` as a prefix.

Setting up a Webots server VM on AWS

Before you start the following steps, make sure that your AWS region is set to `Ohio`. You can change that in the top right corner when you are in your AWS console. You also need a quota of 48 vCPU on g4dn instances to launch a game with 8 robot controllers and one simulator instance (32 vCPU if you have 4 robots and one simulator instance). You can request an increase of your vCPU quota under `Service Quota`.

1. Launch a new instance from a community AMI. Search for `robocup` and use the latest release of the server VM (the official image is published by the AWS account 079967072104).
2. Set `g4dn.4xlarge` as an instance type. Configure the instance by explicitly setting the threads per core from 2 (default) to 1 in the CPU options.
3. Make sure the simulator VM is created in the same virtual private cloud (which is the local network in the cloud) as your client instances.
4. Leave 30 GB of general purpose SSD storage for the instance (as pre-configured in the AMI).

5. In the security group options, make sure that you allow TCP connections for ports 10001, 10002, 10003, 10004, 10021, 10022, 10023, 10024 (for the client API connections to the robot models) as well as UDP connections for port 3838 and 3939 (for the GameController).
6. Once you launch the instance, you can connect to it via ssh using `ssh -i key.pem ubuntu@ip` where `key.pem` is the key you set when creating your instance and ip is the public IP of the instance.
7. Launch an Xorg server by executing the following command:

```
/usr/bin/Xorg -noreset +extension GLX +extension RANDR +extension RENDER \
              -logfile /log-xdummy.log -config /etc/X11/xorg.conf :1
```

8. Configure the following environment variables:
   (a) `export DISPLAY=:1`
   (b) `export GAME_CONTROLLER_HOME=/home/ubuntu/GameController`
   (c) `export JAVA_HOME=/usr`
9. Adapt the `game.json` file in `webots/projects/samples/contests/robocup/controllers/referee`:
   (a) Set `host` to the *local* IP address of your webots server VM.
   (b) For team red and team blue, change the `hosts` list as follows: Keep 127.0.0.1 as the first entry, then add the *local* IP addresses of your client instances as the second to fifth entry.
10. Start the simulation:

```
cd /home/ubuntu/webots
./webots --batch --no-sandbox --no-rendering --stdout --stderr \
         ./projects/samples/contests/robocup/worlds/robocup.wbt
```

If you want to play with less than four robots in a team, you can replace the local IP addresses of your client instances in the `game.json` with 127.0.0.1. Make sure the number of robot controller IP addresses matches the number of robots defined in your `team.json` file.

Docker Images

Teams are expected to provide Docker images for their robot control software via a container registry provided by the Technical Committee. They will receive a specific user account that allows them to read from and write images to the specific container registry for their team. Each Docker image is restricted to a maximum size of 10 GB. The total maximum size of all Docker images in a team's registry is 40 GB. Individual images larger than 10 GB will be automatically deleted from the registry and the game will hence be started without that robot control software. In case that the total maximum size is exceeded, random Docker images are deleted automatically until the total maximum size is below 40 GB. It is possible to configure a different Docker image for each robot in the `team_config.json`. The arguments passed when running a Docker image (the `CMD`) can be configured via the `team_config.json`. Furthermore, there will be the following environment variables available in the Docker containers:

- `ROBOCUP_ROBOT_ID`: The id of your robot as specified in the `team_config.json`. Valid values are 1 - 4 (KidSize) or 1 - 2 (AdultSize).
- `ROBOCUP_TEAM_COLOR`: Your team color as announced in the game schedule. It is equal to either `red` or `blue`.
- `ROBOCUP_SIMULATOR_ADDR`: The IP and port of the player controller for the respective robot - as defined in the game.json. For example, `192.168.123.22:10002`.
- `ROBOCUP_MIRROR_SERVER_IP`: The IP of the mirror server.

  The purpose of the mirror server is to emulate broadcast (broadcast is unavailable): If any of your robot VMs sends data to the mirror server on port 3737, the data will be send ("mirrored") to the other robot VMs of your team on port 3737.
- `ROBOCUP_GAMECONTROLLER_IP`: The IP of the GameController. You can send GameController return messages via UDP port 3939.

- `ROBOCUP_TEAM_PLAYER1_IP`: The IP of the robot VM that runs the first robot of your team ("first" w.r.t. the order of the robots in the team.json.)
- `ROBOCUP_TEAM_PLAYER2_IP`: The IP of the robot VM that runs the second robot of your team
- `ROBOCUP_TEAM_PLAYER3_IP`: The IP of the robot VM that runs the third robot of your team
- `ROBOCUP_TEAM_PLAYER4_IP`: The IP of the robot VM that runs the fourth robot of your team

Uploading Docker Images

The TC will provide teams with:

- AWS Access Key ID
- AWS Secret Access Key
- Docker Registry URL

You can push an image like this:

1. Install the AWS CLI tool: https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
2. Log into the AWS CLI and provide the credentials received from the TC:

   ```
   aws configure
   ```

3. Log into the Docker registry

   ```
   aws ecr get-login-password --region us-east-2 | docker login \
                --username AWS --password-stdin <registry-url>
   ```

   Where `<registry-url>` will be provided by the TC.

   Each registry can exclusively be accessed with the team credentials (and by the TC).

4. Tag your Docker image and push it:

   ```
   docker tag <image> <registry-url>:<optional-tag>
   docker push <registry-url>:<optional-tag>
   ```

   Note that each team only has one Docker image name as defined in the registry URL provided by the TC. However, each team is free to choose the Docker image tags. In the `team.json` file you can configure for each of your robots which Docker image tag is used, see the API specification.

The above commands were only tested under Linux. For more information on pushing Docker images (including how to push Docker images on other operating systems), see https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html

Networking

The network consists of the following participants:

- **Robot VMs:** Each team has 4 (KidSize) or 2 (AdultSize) robot VMs.
- **Simulator VM:**
  - **Webots:** The simulator itself with 4 (KidSize) or 2 (AdultSize) robots per team.
  - **API server:** Robot VMs access their virtual robots via the API server.
  - **GameController:** The GameController sends information about the game state to the robot VMs.
  - **AutoRef:** The AutoRef (Automatic Referee Service) replaces the human referee and communicates its decision to the GameController.
  - **UDPService:** A Python script that emulates broadcast on the network.

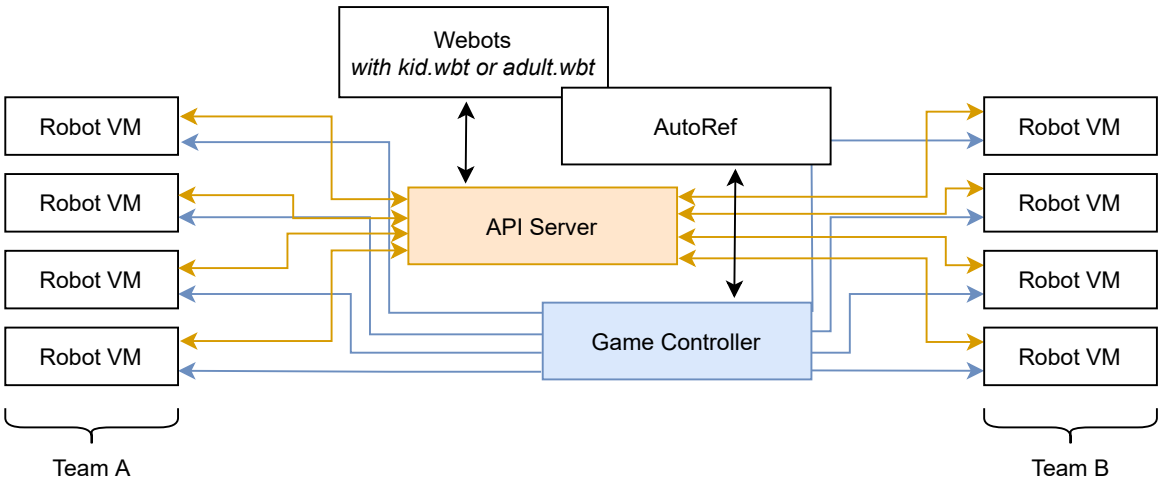

Figure 1: The network communication of the virtual soccer competition. Information from the Webots simulator are only accessible via the API server. The AutoRef only interacts with the GameController and has no direct communication to the robot VMs.

Both the API server and the AutoRef are implemented as Webots supervisors.

The total bandwidth between all robot VMs from a single team and the simulator API is limited to 350 MB/s and is evenly distributed among all robots playing. For example, if a team from KidSize has provided a `team_config.json` with 4 robot ids, each robot VM has a bandwidth of 87.5MB/s. The total bandwidth between robot VMs (of the same team) is limited to 1MB/s. Robot VMs of opposing teams are not allowed not communicate.

The Docker image is run using the host network (think of `docker run --network host`). Thus all ports from within the Docker container will be reachable from outside the robot VM.

For communication between robots, Multicast and Broadcast will not work, which is a limitation of Docker, see [multicast1] and [multicast2] and AWS Virtual Private Clouds, see [broadcast]. Hence, the Technical Committee will provide a UDP-server running on the Simulator VM that handles package sending in the following way:

- **GameController Updates**: As in previous years, robots will receive the GameController state on port `3838`
- **GameController Responses**: As in previous years, teams can send updates to the GameController via port `3939`. Note that this will only transmit the message to the GameController, other robots will *not* receive this message.
- **Team-internal Communication**: To send a message to all robots of your team, you can send a message to port `3737`. Unlike in previous years, the UDP-server ensures that messages sent via this port will only be transmitted to robot VMs of your own team. The UDP server ensures that teams respect the limits on team-internal communication by ignoring all messages that exceed the size limit set in the rule book.

You will receive the IP of the broadcasting service via the `ROBOCUP_MIRROR_SERVER_IP` Docker cmd.

Game Logs

After the video stream of a game is finished, the logs of that game are released. Some logs are released publicly, some are only released to the respective teams.

Public logs:

- **Recording of the game:** The recording will be available both as a video and as a custom format provided by Cyberbotics to generate a 3d-replay version of the game.
- **AutoRef logs:** All decisions taken by the AutoReferee system.
- **GameController logs:** Internal log messages and updates sent to the robot control instances.

Team-internal logs:

- **Docker log:** The output (stdout and stderr) of the robot control software (think of `docker logs`).

- **Custom log folder:** We want to enable teams to record logs for debugging purposes with a limit of 10 GB per robot VM. Thus, each Docker container will have the folder `/robocup-logs` where you are able to write files into. After a game is finished, the files of each of the robot's custom log folder will be transferred to teams. To receive your log files, you will need to provide credentials as a `credentials.json` via the humanoid league submission system for a privately hosted ftp server or an AWS S3 instance before the tournament starts. We will attempt writing the log files as soon as the stream of a game has finished. This folder is not shared across the Docker containers (i.e., you are not able to communicate via this folder with the other Docker containers. Each robot has their own folder.). The folder is also not persistent across games, it will be initially empty at the beginning of each game.

This is an example of an FTP access provided by teams in the `credentials.json`:

```
{
    "type": "sftp",
    "credentials": {
         "hostname": "ab",
         "port": "secret",
         "username": "test_bucket",
         "password": "test_bucket",
         "prefix": "custom_folder_on_your_ftp"
       }

}
```

This is an example of an S3 access provided by teams in the `credentials.json`:

```
{
    "type": "s3",
    "credentials": {
        "access_id": "test_id",
        "access_secret": "test_secret",
        "bucket": "test-bucket",
        "region": "aws-region-of-your-bucket"
    }
}
```

To create an S3 instance, you need to log in to your AWS console and switch to the `S3` dashboard. You can then create a new bucket for the log files. In the `Identity and Access Management (IAM)` dashboard, you can create a new user account to obtain the credentials that need to be provided in the `credentials.json`. We only require programmatic access, no login to the AWS console is required. Once you created the IAM, you will be able to download a file with the access_id and access_secret. You then need to ensure that the newly created user has writing permissions on the bucket and grant it `s3:PutObject` access on the bucket (e.g., as demonstrated in [s3policy]).

Public logs are released to the general public and they will remain accessible at least until the end of the tournament. Team-internal logs will be made available only to the respective team. Team-internal logs are deleted after a successful transfer to the team's ftp or S3 instance or latest after 3 newer versions have come in. Teams are responsible themselves to keep the ftp server or S3 instance available for the TC to transfer data. In case more than three games are waiting to be sent to teams, the oldest one will be deleted irrespective of teams gaining access to them.

Storage

Each VM has a SSD storage with at least 10 GB of free storage. That storage can be accessed via the Docker overlay filesystem (the main filesystem mounted at / in the Docker container). The storage is not persistent across games and is reset after each game.

Note that teams can not pre-populate the filesystem and are expected to store all data they need (including assets such as weights for neural networks) in the Docker image. The filesystem can be only used to store temporary files.

The custom log folder at `/robocup-logs` has an additional 10 GB of storage.

Game Procedure

Prior to the competition, teams submit their `team_config.json` (the team configuration file, see the API document [apidocument]) and PROTO files (the Webots robot models) via the Humanoid League submission system.

During the competition, teams will be informed of the official streaming schedule for each game. Note that the communicated schedule is the *streaming* time and not when the actual simulated games are played. The simulated games will be played in a four hour time frame before the game is streamed for Round Robin matches and a seven hour time frame for knock-out matches.

A game proceeds as follows:

1. Four hours before the official streaming schedule of a round robin match or seven hours before the official streaming start of a knock-out match, the Docker images defined in the `team_config.json` are pulled (they are pulled before each game so teams can make fixes during the competition).
2. The simulation is started by the Technical Committee and the robot model files are loaded into the simulator.
3. Each robot defined in the `team_config.json` is assigned a VM, and on each of these robot VMs, the Docker image (which is already pulled) is run with the command provided in the `team_config.json`.
4. The AutoRef waits at least two minutes between the start of the robot VMs and the start of the match. The start of the match is defined by the game state being set to `READY` for the first half time in accordance with the laws of the game.
5. The GameController announces that the game has ended. The team's Docker containers get a time frame of two minutes for shutting down gracefully before being killed (think of `docker stop --time=120`).
6. The game is streamed on Twitch to the public in accordance with the game schedule.
7. After the game finished streaming, the game logs are transferred to the teams.

Also note that the actual duration of the game will depend on real-time factor of the simulation.

If a team provides an invalid reference to a Docker image, the respective robot model is still spawned in the game in accordance with the `team_config.json`. In case at least one valid Docker image was provided, the game proceeds normally. If no valid Docker image is provided, the game is counted as a forfeit.

References

apidocument API Specifications for Virtual Soccer Competition, https://cdn.robocup.org/hl/wp/2021/05/v-hsc_simulator_api_v1.0.pdf

broadcast AWS Virtual Private Cloud documentation, https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html

multicast1 Cannot receive external multicast inside container, https://github.com/moby/moby/issues/23659

multicast2 Multicast in Overlay driver, https://github.com/moby/libnetwork/issues/552

s3policy Documentation on how to set a policy for a bucket in S3, https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-with-s3-actions.html#using-with-s3-actions-related-to-objects