Journal of Machine Learning Research 22 (2021) 1-5 Submitted 3/20; Revised 7/21; Published 8/21

TensorHive: Management of Exclusive GPU Access for Distributed Machine Learning Workloads

Paweł Rościszewski PAWEL.ROSCISZEWSKI@PG.EDU.PL
Michał Martyniak MICHAL.MARTYNIAK@LINUX.PL
Filip Schodowski FILIPSCHODOWSKI@GMAIL.COM
Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, ul. Gabriela Narutowicza 11/12 80-233 Gdańsk, Poland

Editor: Andreas Mueller

Abstract

TensorHive is a tool for organizing work of research and engineering teams that use servers with GPUs for machine learning workloads. In a comprehensive web interface, it supports reservation of GPUs for exclusive usage, hardware monitoring, as well as configuring, executing and queuing distributed computational jobs. Focusing on easy installation and simple configuration, the tool automatically detects the available computing resources and monitors their utilization. Reservations granted on the basis of flexible access control settings are protected by pluggable violation hooks. The job execution module includes auto-configuration templates for distributed neural network training jobs in frameworks such as TensorFlow and PyTorch. Documentation, source code, usage examples and issue tracking are available at the project page: `https://github.com/roscisz/TensorHive/`

Keywords: GPU reservation, access control, hardware monitoring, job orchestration

1. Background and Motivations

In the face of the spectacular improvements in many practical applications introduced by deep learning (LeCun et al., 2015), both research and engineering teams all over the world are equipped with computing accelerators such as GPUs for neural network training. Because such emerging work groups are often formed for the needs of specific projects with varying requirements and the GPU specifications are rapidly changing, we argue that there is an increasing need for GPU management approaches that are easy to set up and tailored for contemporary machine learning workloads. At the same time, quality of the trained models depends heavily on the sizes of training data sets and trained models, as well as experimental optimization of multiple hyper-parameters (Hestness et al., 2017). This makes model training workloads heavy enough to consider speeding them up by combining computing power of multiple accelerators using data and model parallelism methods (Dean et al., 2012). Distributed implementations based on deep learning frameworks such as TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019) require new job orchestration tools.

In *high performance computing* (HPC), sharing computing resources between multiple users is often organized using batch queuing systems such as SLURM, an open-source, highly scalable job scheduling system for Linux clusters proposed by Yoo et al. (2003). In such systems, access to resources is granted to users for some duration of time, based on certain

priorities and limits. Users submit their computational tasks to queues for execution in the future. Although this has been the usual use case in HPC systems, we would like to point out the importance of another one, which we find useful for the aforementioned machine learning work groups: reserving in advance some duration of time when exclusive access to selected GPUs is granted to the user. This way, not only workloads prepared beforehand can be executed, but also interactive work is supported, including experimentation and iterative code development. This is often desirable in neural network training applications, because it allows for debugging, babysitting the models, tuning the training programs to the capabilities of the specific GPUs, testing various training parameters such as batch size, model modifications, data augmentation modes, etc.

Resource allocation for immediate interactive work is possible in SLURM using *salloc*, but reservation for future is only available for super users that have access to the *scontrol* command. SLURM's modular design and flexible plugin mechanism has allowed to introduce multiple features and tools, such as generic resource scheduling[1] that allows to manage GPU allocation or the "SLURM-web" web application frontend[2]. However, although simplicity was one of the main design goals of Yoo et al. (2003), using the additional plugins and tools developed over the years complicates the installation and configuration process.

Some overhead has also to be added to the basic administration effort (significant itself) required to enable convenient running of distributed trainings on platforms specialized for other use cases. For example, *tf-operator*[3] for the container orchestration system Kubernetes (Burns et al., 2016) or *tf-yarn*[4] for the big data computing resource negotiator Apache Hadoop YARN (Vavilapalli et al. 2013). Our motivation is to develop a standalone solution tailored for typical model training scenarios, that would work out of the box.

In this paper we propose TensorHive, an open-source GPU management tool that combines exclusive GPU reservations with job execution and monitoring, while focusing on user-friendliness, simple configuration and support for frameworks and scenarios typical for high performance machine learning. We describe key features of the tool in Section 2, a technical overview in Section 3 and we summarize the paper in Section 4.

2. Key Features

The key features of TensorHive can be divided into three categories: *reservations*, *job execution* and *monitoring* described in Sections 2.1, 2.2 and 2.3, respectively.

2.1 Reservations

Long term overview: A table with long-term overview of GPU occupancy. Available GPUs grouped by host constitute the table rows, while columns represent consecutive hours in a chosen time span. Cell colors represent the reservation status (free, reserved by someone else, reserved by the user). The overview is useful for narrowing the set of considered GPUs, especially when the number of available GPUs is too high for detailed visualization. Hosts or GPUs selected in this view will be shown in the *reservation calendar*.

1. See `https://slurm.schedmd.com/gres.html`.
2. See `https://github.com/edf-hpc/slurm-web`.
3. See `https://github.com/kubeflow/tf-operator`.
4. See `https://github.com/criteo/tf-yarn`.

Reservation calendar: A calendar widget focused on chosen 7 days. Sub-columns for each day, corresponding to the consecutive GPUs, are filled with boxes containing reservation owner's usernames and short reservation descriptions. New reservations can be made by interacting with the calendar widget, and specifying the details in a dialog panel. Overriding existing reservations is forbidden, cancellation is available to owners and superusers.
Access control: Administrators can grant specific users (or groups) access to specific devices (or nodes), in flexibly configurable time windows (e.g. Mondays 8AM-4PM).
Violation handlers: The TensorHive core program actively monitors the processes that use each GPU. If one of them belongs to a user who does not own the current reservation, configurable violation handlers are called, including informing the offender via interactive terminals and sending appropriate e-mail to the offender and system admin.
Usage statistics: Among the details of already outdated reservations, usage statistics are available, such as average GPU utilization throughout the reservation time.

2.2 Job Execution

Definition: Jobs in TensorHive are coherently managed groups of tasks - shell commands composed from interlaced static and variable fields. The latter can be used for conveniently duplicating commands and changing only certain parameters. Assigning tasks to specific devices results in executing them on the appropriate hosts and setting the appropriate value of the CUDA_VISIBLE_DEVICES environment variable.
Queue for non-reserved time: Users can add jobs to a queue, so that they will be automatically executed by TensorHive when it detects a longer period when all resources required by the job tasks are not reserved.
Templates: Defining training tasks for chosen frameworks is vastly simplified by templates that automatically configure cluster definition arguments, including TF_CONFIG and standard command-line parameters for TensorFlow and PyTorch. Usage examples for each template are provided in the `examples/` directory.
Attaching to interactive process sessions: Each process is executed within *GNU screen*. This allows for example to gain interactive control over the process by logging to the appropriate host and attaching to the process session using the `screen -x` option.

2.3 Monitoring

A re-arrangable chart dashboard presents the following parameters of auto-detected devices:
GPU and CPU state: Time series charts show recent values of various GPU and CPU attributes, including utilization and memory usage.
GPU process monitoring: Processes that currently use GPUs on the selected host are listed in a table with owner usernames, PIDs and executed commands in columns.

3. Technical Overview

Essentially, TensorHive is a Python program that maintains SSH connections to a set of hosts and provides a REST API for convenient interaction with them. Although the API could be useful via various kinds of user interfaces (for example command-line scripts using curl), the tool provides also a dedicated web application in Vue.js, self-hosted by default.

Supported platforms: TensorHive is a package for Python v3.5+, developed and tested on various distributions of GNU/Linux including Debian Buster and Ubuntu 18.04, both on the hosting and the monitored nodes. For connectivity, TensorHive requires that SSH connections are possible from the appropriate user on the hosting node to the monitored nodes without password authentication. We suggest key-based authentication, which is often already configured in target environments. For GPU monitoring, *nvidia-smi* is required.
Installation: Current version with all dependencies can be installed using `pip install tensorhive`. For a specific version of the repository directory `pip install .` can be used.
Configuration and running: By default, configuration files will be installed in the home configuration directory: `~/.config/TensorHive`. Minimal configuration requires only providing hostnames and usernames for logging into the available computing nodes in hosts_config.json. Advanced options, including listening URLs, paths and violation handler settings, can be found in main_config.json, while e-mail settings for warnings in mailbot_config.json. The tool is launched simply by executing the `tensorhive` command, which also provides additional options (see `tensorhive --help` for usage and `tensorhive init` for semi-automatic configuration wizard that is automatically started in the first run).
Advanced authorization for job execution: Users who wish to use the job execution feature need to authorize the tool to log in to their respective accounts via SSH. Authorization can be easily granted or revoked by adding/removing the public TensorHive key to .ssh/authorized_keys on the appropriate hosts. The public key is available through the *tensorhive key* command and on the web application login page.
Development: TensorHive is distributed under the Apache Licence 2.0, versioned under a git repository, following the git-flow pattern, hosted on GitHub. We use GitHub issue tracking tools for bug and feature reports, as well as code review through pull requests. Code quality is inspected by the flake8 and mypy tools, and chosen parts of the core code are subject to unit tests during build checks on continuous integration platforms.

4. Summary

TensorHive is an open-source tool that facilitates efficient coordination of exclusive access to computing resources for machine learning workloads by introducing a hybrid mechanism of reservations and job queuing. The monitoring and job execution modules along with reservation violation handlers encourage the users to fully utilize the granted resources, while the flexible access control rules allow the admins to look after proper resource usage.

Acknowledgments

This work has been partially supported by Statutory Funds of Electronics, Telecommunications and Informatics Faculty, Gdańsk University of Technology that provided the NVIDIA DGX servers for deployment tests. The work was partially supported within grant funded by the European Union Regional Operation Program for the Pomeranian Voivodship for years 2014-2020 based on the Resolution no 190/214/17 of the Pomeranian Voivodship Board. The authors would like to thank the top TensorHive contributors: Bartosz Jankowski, Tomasz Menet, Martyna Oleszkiewicz, Mateusz Piotrowski and Jacek Szempliński.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Commun. ACM*, 59(5):50–57, 2016. URL `http://doi.acm.org/10.1145/2890784`.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and others. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012. URL `http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks`.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Patwary, Mostofa Ali, Yang Yang, and Yanqi Zhou. Deep Learning Scaling is Predictable, Empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, May 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539. URL `http://www.nature.com/doifinder/10.1038/nature14539`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Vinod Kumar Vavilapalli, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, Eric Baldeschwieler, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, and Hitesh Shah. Apache Hadoop YARN: yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing - SOCC '13*, pages 1–16, Santa Clara, California, 2013. ACM Press. ISBN 978-1-4503-2428-1. doi: 10.1145/2523616.2523633. URL `http://dl.acm.org/citation.cfm?doid=2523616.2523633`.

Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, pages 44–60, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39727-4. doi: 10.1007/10968987_3.