Optimale Cluster- und GPU-Auslastung bei AI-Ausführung

NetApp Solutions

NetApp
April 25, 2024

Inhalt

Optimale Cluster- und GPU-Auslastung bei Run:AI

In den folgenden Abschnitten finden Sie Einzelheiten zum Durchlauf:AI-Installation, Testszenarien und Ergebnisse, die bei dieser Validierung durchgeführt wurden.

Betrieb und Performance dieses Systems haben wir mithilfe von Industriestandard-Benchmark-Tools, einschließlich TensorFlow Benchmarks, validiert. Mit dem ImageNet-Datensatz wurde ResNet-50 trainiert, ein berühmtes Convolutional Neural Network (CNN) DL-Modell für die Bildklassifizierung. ResNet-50 liefert ein präzises Trainingsergebnis mit einer schnelleren Verarbeitungszeit, wodurch wir eine ausreichende Nachfrage nach dem Storage-System erhöhen können.

Run:AI Installation

So installieren Sie Run:AI:

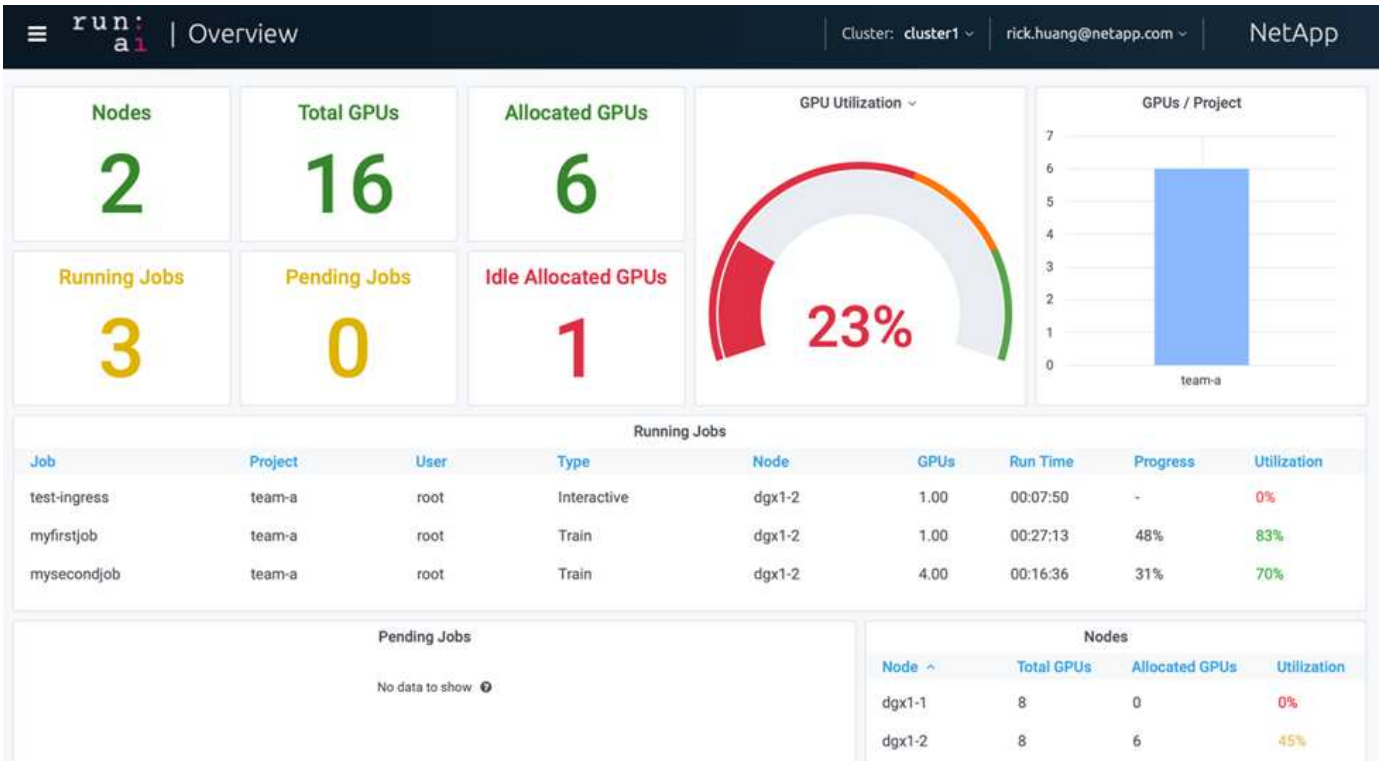
1. Installieren Sie den Kubernetes-Cluster mit DeepOps und konfigurieren Sie die NetApp Standard-Storage-Klasse.
2. GPU-Nodes vorbereiten:
   a. Vergewissern Sie sich, dass NVIDIA-Treiber auf GPU-Nodes installiert sind.
   b. Verifizieren Sie das `nvidia-docker` Ist als Standard-Docker-Laufzeit installiert und konfiguriert.
3. Install-Run:KI:
   a. Melden Sie sich bei an "Ausführung:KI-Admin-UI" Um den Cluster zu erstellen.
   b. Laden Sie das erstellte herunter `runai-operator-<clustername>.yaml` Datei:
   c. Wenden Sie die Bedienerkonfiguration auf das Kubernetes-Cluster an.

```
kubectl apply -f runai-operator-<clustername>.yaml
```

4. Überprüfen Sie die Installation:
   a. Gehen Sie zu "https://app.run.ai/".
   b. Wechseln Sie zum Dashboard „Übersicht“.
   c. Überprüfen Sie, ob die Anzahl der GPUs oben rechts die erwartete Anzahl von GPUs und die GPU-Nodes alle in der Liste der Server vorhanden sind.Weitere Informationen zu Run:KI-Implementierung finden Sie unter "Installieren von Run:AI in einem lokalen Kubernetes-Cluster" Und "Installieren der Run:AI-CLI".

Run:AI Dashboards und Ansichten

Nach der Installation von Run:AI auf dem Kubernetes-Cluster und der korrekten Konfiguration der Container werden die folgenden Dashboards und Ansichten auf angezeigt "https://app.run.ai" Im Browser, wie in der folgenden Abbildung dargestellt.



Es gibt insgesamt 16 GPUs im Cluster, die von zwei DGX-1-Nodes bereitgestellt werden. Sie sehen die Anzahl der Nodes, die insgesamt verfügbaren GPUs, die zugewiesenen GPUs, die Workloads zugewiesen werden, die Gesamtzahl der ausgeführten Jobs, ausstehende Jobs und inaktive GPUs. Rechts im Balkendiagramm werden GPUs pro Projekt angezeigt, die die Verwendung der Cluster-Ressource durch die verschiedenen Teams zusammenfassen. In der Mitte befindet sich die Liste der aktuell ausgeführten Jobs mit Jobdetails, einschließlich Jobname, Projekt, Benutzer, Jobtyp, Der Node, auf dem jeder Job ausgeführt wird, die Anzahl der für diesen Job zugewiesenen GPU(s), die aktuelle Laufzeit des Jobs, der Job-Fortschritt in Prozent und die GPU-Auslastung für diesen Job. Beachten Sie, dass das Cluster nicht ausgelastet ist (GPU-Auslastung bei 23 %), da nur drei laufende Jobs von einem einzigen Team eingereicht werden (`team-a`).

Im folgenden Abschnitt zeigen wir, wie auf der Registerkarte „Projekte“ mehrere Teams erstellt und jedem Team GPUs zugewiesen werden können, um die Cluster-Auslastung zu maximieren und Ressourcen zu managen, wenn pro Cluster mehrere Benutzer vorhanden sind. Die Testszenarien nachahmen Enterprise-Umgebungen, in denen Speicher- und GPU-Ressourcen unter Training, Inferenz und interaktiven Workloads gemeinsam genutzt werden.

Projekte für Data Science-Teams erstellen und GPUs zuweisen

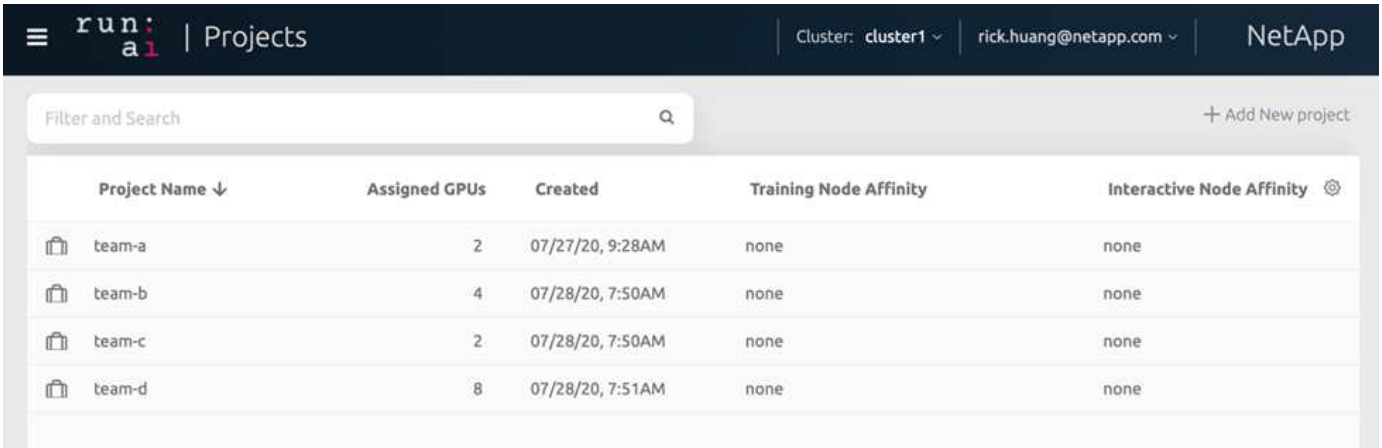
Forscher können Workloads über Run:AI CLI, Kubeflow oder ähnliche Prozesse senden. Um die Ressourcenzuweisung zu optimieren und Priorisierungen zu erstellen, führt Run:AI das Projektkonzept ein. Projekte sind Quoteneinheiten, die einen Projektnamen mit GPU-Zuweisung und -Einstellungen verknüpfen. Mehrere Data-Science-Teams können auf einfache und bequeme Weise gemanagt werden.

Ein Forscher, der einen Workload einreicht, muss ein Projekt mit einer Workload-Anforderung verknüpfen. Der Scheduler:AI vergleicht die Anforderung mit den aktuellen Zuweisungen und dem Projekt und bestimmt, ob der Workload Ressourcen zugewiesen werden kann oder ob er sich im ausstehenden Status befindet.

Als Systemadministrator können Sie auf der Registerkarte Run:AI Projects die folgenden Parameter einstellen:

- **Modellprojekte.** ein Projekt pro Benutzer festlegen, ein Projekt pro Benutzerteam festlegen und ein Projekt für ein echtes organisatorisches Projekt festlegen.
- **Projektquoten.** jedes Projekt ist mit einer Quote von GPUs verknüpft, die für dieses Projekt gleichzeitig zugewiesen werden können. Dies ist eine garantierte Quote, da Forscher, die dieses Projekt nutzen, garantiert sind, diese Anzahl von GPUs zu erhalten, egal wie der Status im Cluster ist. In der Regel sollte die Summe der Projektzuweisung der Anzahl der GPUs im Cluster entsprechen. Darüber hinaus kann ein Benutzer dieses Projekts eine Überkontingente erhalten. Solange GPUs nicht verwendet werden, kann ein Forscher, der dieses Projekt verwendet, mehr GPUs erhalten. Wir führen Testszenarien mit Überquoten und fairer Erwägungen ein "Erreichen einer hohen Cluster-Auslastung mit GPU-Zuweisung über ein Kontingent", "Gerechtigkeit Bei Der Grundlegenden Ressourcenzuweisung", und "Gerechtigkeit Wegen Zu Viel Quoten".
- Erstellen Sie ein neues Projekt, aktualisieren Sie ein vorhandenes Projekt und löschen Sie ein vorhandenes Projekt.
- **Anzahl der Aufträge, die auf bestimmten Knotengruppen ausgeführt werden sollen**. Sie können bestimmte Projekte nur auf bestimmten Knoten ausführen. Dies ist nützlich, wenn das Projektteam spezielle Hardware benötigt, zum Beispiel mit genügend Arbeitsspeicher. Alternativ kann ein Projektteam Eigentümer bestimmter Hardware sein, die mit einem speziellen Budget erworben wurde, oder wenn Unternehmen direkte Build- oder interaktive Workloads für die Arbeit an schwächerer Hardware und das direkte Training oder unbeaufsichtigte Arbeitslasten auf schnellere Nodes benötigen. Informationen zu Befehlen zum Gruppieren von Knoten und zum Festlegen der Affinität für ein bestimmtes Projekt finden Sie im "Ausführen:KI-Dokumentation".
- **Beschränken Sie die Dauer von interaktiven Jobs**. Forscher vergessen häufig, interaktive Jobs zu schließen. Dies könnte zu einer Verschwendung von Ressourcen führen. Einige Organisationen ziehen es vor, die Dauer von interaktiven Jobs zu begrenzen und automatisch zu schließen.

Die folgende Abbildung zeigt die Ansicht „Projekte“ mit vier erstellten Teams. Jedem Team wird eine unterschiedliche Anzahl von GPUs zugewiesen, die verschiedenen Workloads Rechnung tragen. Die Gesamtzahl der GPUs entspricht der Gesamtzahl der verfügbaren GPUs in einem Cluster, der aus zwei DGX-1 besteht.



Senden von Jobs in Run:AI CLI

Dieser Abschnitt enthält die Details zum grundlegenden Befehl „Run:AI“, mit denen Sie jeden Kubernetes-Job ausführen können. Je nach Workload-Typ in drei Teile unterteilt: KI/ML/DL-Workloads können in zwei generische Typen unterteilt werden:

- **Nicht besuchte Schulungen**. Bei diesen Workloads bereitet der Data Scientist einen selbstständigen Workload auf und sendet ihn zur Ausführung. Während der Durchführung kann der Kunde die Ergebnisse untersuchen. Dieser Workload wird häufig in der Produktion verwendet oder wenn sich die Modellentwicklung zu einer Phase befindet, in der kein menschliches Eingreifen erforderlich ist.
- **Interaktive Build-Sitzungen**. Bei diesen Workloads öffnet der Data Scientist eine interaktive Sitzung mit Bash, Jupyter Notebook, Remote PyCharm oder ähnlichen IDEs und greift direkt auf GPU-Ressourcen zu. Wir fügen ein drittes Szenario für die Ausführung interaktiver Workloads mit angeschlossenen Ports ein, um dem Container-Benutzer einen internen Port zu zeigen.

Unbeaufsichtigte Trainings-Workloads

Nachdem Sie Projekte eingerichtet und GPU(s) zugewiesen haben, können Sie jeden beliebigen Kubernetes-Workload mit dem folgenden Befehl an der Befehlszeile ausführen:

```
$ runai project set team-a runai submit hyper1 -i gcr.io/run-ai-
demo/quickstart -g 1
```

Mit diesem Befehl wird ein unbeaufsichtigter Schulungsauftrag für Team A mit einer Zuweisung einer einzelnen GPU gestartet. Der Job basiert auf einem Beispielbild für Docker. `gcr.io/run-ai-demo/quickstart`. Wir haben die Aufgabe genannt `hyper1`. Sie können dann den Fortschritt des Jobs überwachen, indem Sie folgenden Befehl ausführen:

```
$ runai list
```

Die folgende Abbildung zeigt das Ergebnis des `runai list` Befehl. Folgende typische Status werden möglicherweise angezeigt:

- `ContainerCreating`. Der Docker-Container wird aus dem Cloud-Repository heruntergeladen.
- `Pending`. Der Job wartet auf die Planung.
- `Running`. Der Job wird ausgeführt.

```
~> runai list
Showing jobs for project team-a
NAME    STATUS   AGE  NODE                                    IMAGE                        TYPE         PROJECT  USER   GPUs
hyper1  Running  11s  gke-dev-yaron1-gpu-4-pool-154f511d-5nk5  gcr.io/run-ai-demo/quickstart  Train        team-a   yaron  1
```

Führen Sie den folgenden Befehl aus, um einen zusätzlichen Status für Ihren Job zu erhalten:

```
$ runai get hyper1
```

Um die Protokolle des Jobs anzuzeigen, führen Sie den aus `runai logs <job-name>` Befehl:

```
$ runai logs hyper1
```

In diesem Beispiel sollten Sie das Protokoll einer laufenden DL-Sitzung sehen, einschließlich der aktuellen

Trainingszeit, ETA, Wert der Verlustfunktion, Genauigkeit und Zeit, die für jeden Schritt verstrichen ist.

Der Cluster-Status kann in der Benutzeroberfläche „Run:AI“ von angezeigt werden "https://app.run.ai/". Unter Dashboards > Übersicht können Sie die GPU-Auslastung überwachen.

Um diesen Workload zu beenden, führen Sie den folgenden Befehl aus:

```
$ runai delte hyper1
```

Dieser Befehl stoppt den Trainings-Workload. Sie können diese Aktion überprüfen, indem Sie ausführen `runai list` Ein weiteres Jahr in der Weitere Informationen finden Sie unter "Unbeaufsichtigte Trainings-Workloads starten".

Interaktive Build-Workloads

Nach dem Einrichten von Projekten und der Zuweisung von GPU(s) können Sie einen interaktiven Build-Workload mit dem folgenden Befehl in der Befehlszeile ausführen:

```
$ runai submit build1 -i python -g 1 --interactive --command sleep --args
infinity
```

Der Job basiert auf einer Beispieldatei-Bildpython. Wir nannten den Job build1.

> Der `-- interactive` Flag bedeutet, dass der Job kein Start oder Ende hat Es liegt in der Verantwortung des Forschers, den Job zu schließen. Der Administrator kann ein Zeitlimit für interaktive Jobs festlegen, nach denen sie vom System beendet werden.

Der `--g 1` Flag weist diesem Job eine einzelne GPU zu. Der Befehl und das angegebene Argument lautet `--command sleep—args infinity`. Sie müssen einen Befehl angeben, oder der Container startet und wird sofort beendet.

Die folgenden Befehle funktionieren ähnlich wie die in beschriebenen Befehle Unbeaufsichtigte Trainings-Workloads:

- `runai list`: Zeigt den Namen, Status, Alter, Knoten, Bild, Projekt, Benutzer und GPUs für Jobs.
- `runai get build1`: Zeigt zusätzlichen Status im Job build1 an.
- `runai delete build1`: Stoppt die interaktive Arbeitslast build1.um eine Bash-Shell in den Container zu bekommen, den folgenden Befehl:

```
$ runai bash build1
```

Dadurch wird eine direkte Shell in den Computer integriert. Data Scientists können dann ihre Modelle innerhalb des Containers entwickeln oder feinfinden.

Der Cluster-Status kann in der Benutzeroberfläche „Run:AI“ von angezeigt werden "https://app.run.ai". Weitere Informationen finden Sie unter "Starten und Verwenden interaktiver Build-Workloads".

Interaktive Workloads mit verbundenen Ports

Als Erweiterung von interaktiven Build-Workloads können Sie dem Container-Benutzer interne Ports beim Starten eines Containers mit der Run:AI-CLI offenbaren. Dies ist nützlich für Cloud-Umgebungen, die Arbeit mit Jupyter Notebooks oder die Verbindung zu anderen Microservices. "Eindringen" Zugriff auf Kubernetes-Services von außerhalb des Kubernetes-Clusters aus. Sie können den Zugriff konfigurieren, indem Sie eine Sammlung von Regeln erstellen, die definieren, welche eingehenden Verbindungen welche Dienste erreichen.

Für eine bessere Verwaltung des externen Zugriffs auf die Services in einem Cluster empfehlen wir, Cluster-Administratoren zu installieren "Eindringen" Und konfigurieren den Load Balancer.

Um Ingress als Servicetyp zu verwenden, führen Sie den folgenden Befehl aus, um den Methodentyp und die Ports beim Senden des Workloads festzulegen:

```
$ runai submit test-ingress -i jupyter/base-notebook -g 1 \
  --interactive --service-type=ingress --port 8888 \
  --args="--NotebookApp.base_url=test-ingress" --command=start-notebook.sh
```

Nachdem der Container erfolgreich gestartet wurde, führen Sie die Ausführung aus `runai list` Um den anzuzeigen `SERVICE URL(S)` Mit dem auf das Jupyter Notebook zugegriffen werden kann. Die URL setzt sich aus dem Ingress-Endpunkt, dem Job-Namen und dem Port zusammen. Siehe zum Beispiel https://10.255.174.13/test-ingress-8888.

Weitere Informationen finden Sie unter "Starten eines interaktiven Build-Workloads mit verbundenen Ports".

Erreichen Einer Hohen Cluster-Auslastung

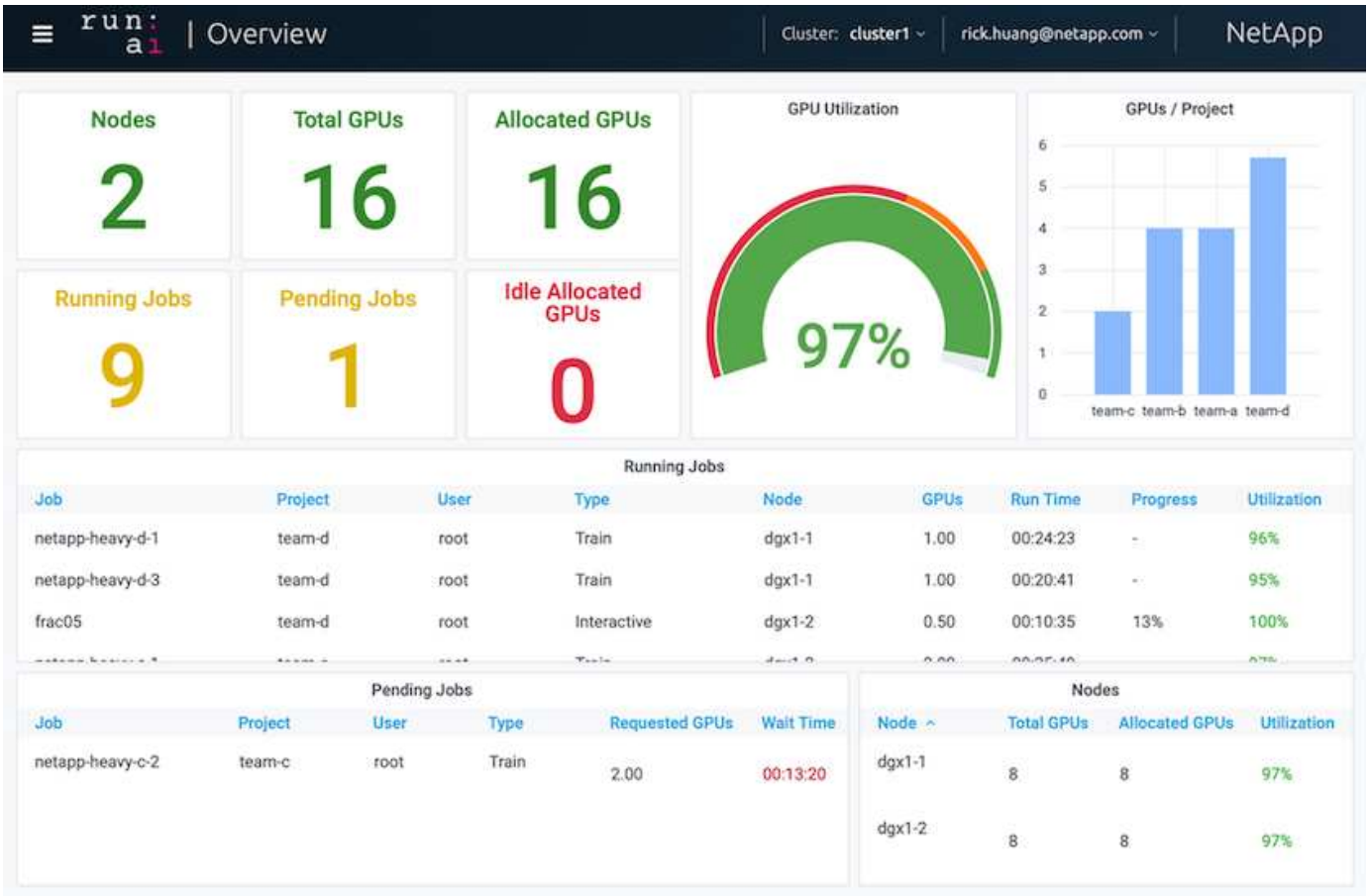
In diesem Abschnitt emulieren wir ein realistisches Szenario, in dem vier Data-Science-Teams jeweils ihre eigenen Workloads einreichen, um die Run:KI-Orchestrierungslösung vorzuführen. Diese Lösung erzielt eine hohe Cluster-Auslastung, während Priorisierung beibehalten und GPU-Ressourcen ausgeglichen werden. Wir beginnen mit dem im Abschnitt beschriebenen ResNet-50-Benchmark "ResNet-50 mit ImageNet-Datensatz Benchmark-Zusammenfassung":

```
$ runai submit netapp1 -i netapp/tensorflow-tf1-py3:20.01.0 --local-image
--large-shm  -v /mnt:/mnt -v /tmp:/tmp --command python --args
"/netapp/scripts/run.py" --args "--
dataset_dir=/mnt/mount_0/dataset/imagenet/imagenet_original/" --args "--
num_mounts=2"  --args "--dgx_version=dgx1" --args "--num_devices=1" -g 1
```

Wir führten den gleichen ResNet-50-Benchmark wie in aus "NVA-1121". Wir haben die Flagge benutzt `--local-image` Für Container, die sich nicht im öffentlichen Docker-Repository befinden. Wir haben die Verzeichnisse angehängt `/mnt` Und `/tmp` Auf dem Host DGX-1-Node zu `/mnt` Und `/tmp` Zum Behälter bzw.. Der Datensatz befindet sich bei NetApp AFFA800 mit dem `dataset_dir` Argument, das auf das Verzeichnis verweist. Beides `--num_devices=1` Und `-g 1` Meinen, dass wir eine GPU für diesen Job zuweisen. Ersteres ist ein Argument für das `run.py` Skript, während letzteres eine Flagge für die ist `runai submit` Befehl.

Die folgende Abbildung zeigt ein Dashboard mit Systemübersicht mit einer GPU-Auslastung von 97 % und

allen sechzehn verfügbaren GPUs. Im Balkendiagramm der GPUs können Sie leicht sehen, wie viele GPUs jedem Team zugewiesen sind. Im Bereich laufende Jobs werden die aktuell ausgeführten Jobnamen, Projekte, Benutzer, Typ, Knoten, angezeigt. GPUs – verbrauchte, Laufzeit, Fortschritt und Auslastungsdetails Eine Liste der Workloads in der Warteschlange mit deren Wartezeit wird unter Ausstehende Jobs angezeigt. Schließlich bietet die Box Nodes GPU-Nummern und Auslastung für einzelne DGX-1-Nodes im Cluster.



Zuweisung von fraktionalen GPUs für weniger anspruchsvolle oder interaktive Workloads

Wenn Forscher und Entwickler an ihren Modellen arbeiten, sei es in der Entwicklung, im Hyperparameter-Tuning oder in der Debugging-Phase, sind für solche Workloads in der Regel weniger Computing-Ressourcen erforderlich. Daher ist es effizienter, fraktionale GPU und Speicher so bereitzustellen, dass dieselbe GPU gleichzeitig anderen Workloads zugewiesen werden kann. Run:die KI-Orchestrierungslösung bietet ein fraktionaler GPU-Sharing-System für Container-Workloads auf Kubernetes. Das System unterstützt Workloads mit CUDA-Programmen und eignet sich insbesondere für schlanke KI-Aufgaben wie Inferenz und Modellbau. Das fraktionale GPU-System ermöglicht Data-Science- und KI-Engineering-Teams die gleichzeitige Ausführung mehrerer Workloads auf einer einzelnen GPU. Auf diese Weise können Unternehmen mehr Workloads ausführen, wie Computervision, Spracherkennung und natürliche Sprachverarbeitung auf derselben Hardware, wodurch sich die Kosten senken lassen.

Run:das fraktionale GPU-System von AI erstellt effektiv virtualisierte logische GPUs mit eigenem Speicher und

Computerraum, den Container nutzen und so zugreifen können, als wären sie eigenständige Prozessoren. So können mehrere Workloads in Containern parallel auf derselben GPU ausgeführt werden, ohne dass sie sich gegenseitig stören. Die Lösung ist transparent, einfach und tragbar und erfordert keine Änderungen an den Containern selbst.

Eine typische Usease könnte zwei bis acht Jobs auf derselben GPU sehen, was bedeutet, dass Sie die achtfache Arbeit mit der gleichen Hardware erledigen können.

Für den Job `frac05` Zu Projekt gehören `team-d` Die folgende Abbildung zeigt, dass die Anzahl der zugewiesenen GPUs 0.50 war. Dies wird weiter durch die überprüft `nvidia-smi` Befehl, der zeigt, dass der für den Container verfügbare GPU-Speicher 16,255 MB betrug: Die Hälfte der 32 GB pro V100 GPU im DGX-1-Node.

```
root@run-deploy:~# runai bash frac05 -p team-d
root@frac05-0:/workload# nvidia-smi
Tue Jul 28 15:17:03 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.05    Driver Version: 450.51.05    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:07:00.0 Off |                    0 |
| N/A   57C    P0   240W / 300W |  15525MiB / 16255MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       156      C   python3                       15525MiB |
+-----------------------------------------------------------------------------+
```

Erreichen einer hohen Cluster-Auslastung mit GPU-Zuweisung über ein Kontingent

In diesem Abschnitt und in den Abschnitten "Gerechtigkeit Bei Der Grundlegenden Ressourcenzuweisung", und "Gerechtigkeit Wegen Zu Viel Quoten", Wir haben erweiterte Testszenarien entwickelt, um die Funktionen der Run:KI-Orchestrierung für komplexes Workload-Management, automatisches präventiv Planen und GPU-Bereitstellung über Kontingente zu demonstrieren. Ziel war es, eine hohe Cluster-Ressourcen-Auslastung zu erzielen und die Produktivität des Data Science-Teams der Enterprise-Klasse in einer ONTAP KI-Umgebung zu optimieren.

Legen Sie für diese drei Abschnitte die folgenden Projekte und Quoten fest:

Projekt	Kontingente
Team A	4
Team b	2
Team c	2
Team d	8

Zusätzlich werden für diese drei Abschnitte folgende Container verwendet:

- Jupyter Notebook: `jupyter/base-notebook`
- Run:AI quickstart: `gcr.io/run-ai-demo/quickstart`

Für dieses Testszenario setzen wir folgende Ziele:

- Zeigen Sie die Einfachheit der Ressourcenbereitstellung und wie Ressourcen von Benutzern abstrahiert werden
- Zeigen Sie, wie Benutzer Fraktionen einer GPU und einer ganzen Zahl von GPUs einfach bereitstellen können
- Zeigen Sie, wie das System Compute-Engpässe beseitigt, indem es Teams oder Benutzern ermöglicht, ihre Ressourcenkontingente zu überziehen, wenn es kostenlose GPUs im Cluster gibt
- Zeigen Sie, wie Engpässe bei der Datenpipeline mithilfe der NetApp Lösung bei rechenintensiven Aufgaben wie dem NetApp Container beseitigt werden
- Darstellung der Ausführung mehrerer Container mithilfe des Systems
  - Jupyter Notebook
  - Run:AI Container
- Zeigt eine hohe Auslastung an, wenn das Cluster voll ist

Einzelheiten zur tatsächlichen Befehlsfolge, die während des Tests ausgeführt wurde, finden Sie unter "Testdetails für Abschnitt 4.8".

Wenn alle 13 Workloads übermittelt werden, wird eine Liste der zugewiesenen Containernamen und -GPUs angezeigt, wie in der folgenden Abbildung dargestellt. Wir haben sieben Trainings- und sechs interaktive Jobs und simulieren dabei vier Data-Science-Teams, die jeweils über eigene Modelle im Einsatz oder in der Entwicklung verfügen. Bei interaktiven Jobs verwenden einzelne Entwickler Jupyter Notebooks, um ihren Code zu schreiben oder zu debuggen. Dadurch eignet es sich, GPU-Fraktionen ohne zu viele Cluster-Ressourcen bereitzustellen.

```
root@run-deploy:~# runai list -A
NAME           STATUS   AGE  NODE    IMAGE                          TYPE        PROJECT  USER  GPUs  CREATED BY CLI  SERVICE URL(S)
b-4-gg         Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-b   root  2     true
c-5-g          Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-c   root  1     true
c-4-gg         Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-c   root  2     true
b-3-g          Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-b   root  1     true
c-3-g02        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.2   true
d-1-gggg       Running  2m   dgx1-2  gcr.io/run-ai-demo/quickstart  Train       team-d   root  4     true
c-2-g03        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.3   true
c-1-g05        Running  2m   dgx1-1  gcr.io/run-ai-demo/quickstart  Interactive team-c   root  0.5   true
a-2-gg         Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-a   root  2     true
b-2-g04        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive team-b   root  0.4   true
a-1-g          Running  3m   dgx1-1  gcr.io/run-ai-demo/quickstart  Train       team-a   root  1     true
b-1-g06        Running  3m   dgx1-2  gcr.io/run-ai-demo/quickstart  Interactive team-b   root  0.6   true
a-1-1-jupyter  Running  3m   dgx1-1  jupyter/base-notebook          Interactive team-a   root  1     true            http://10.61.218.134/a-1-1-jupyter,
https://10.61.218.134/a-1-1-jupyter
```

Die Ergebnisse dieses Testszenarios zeigen Folgendes:

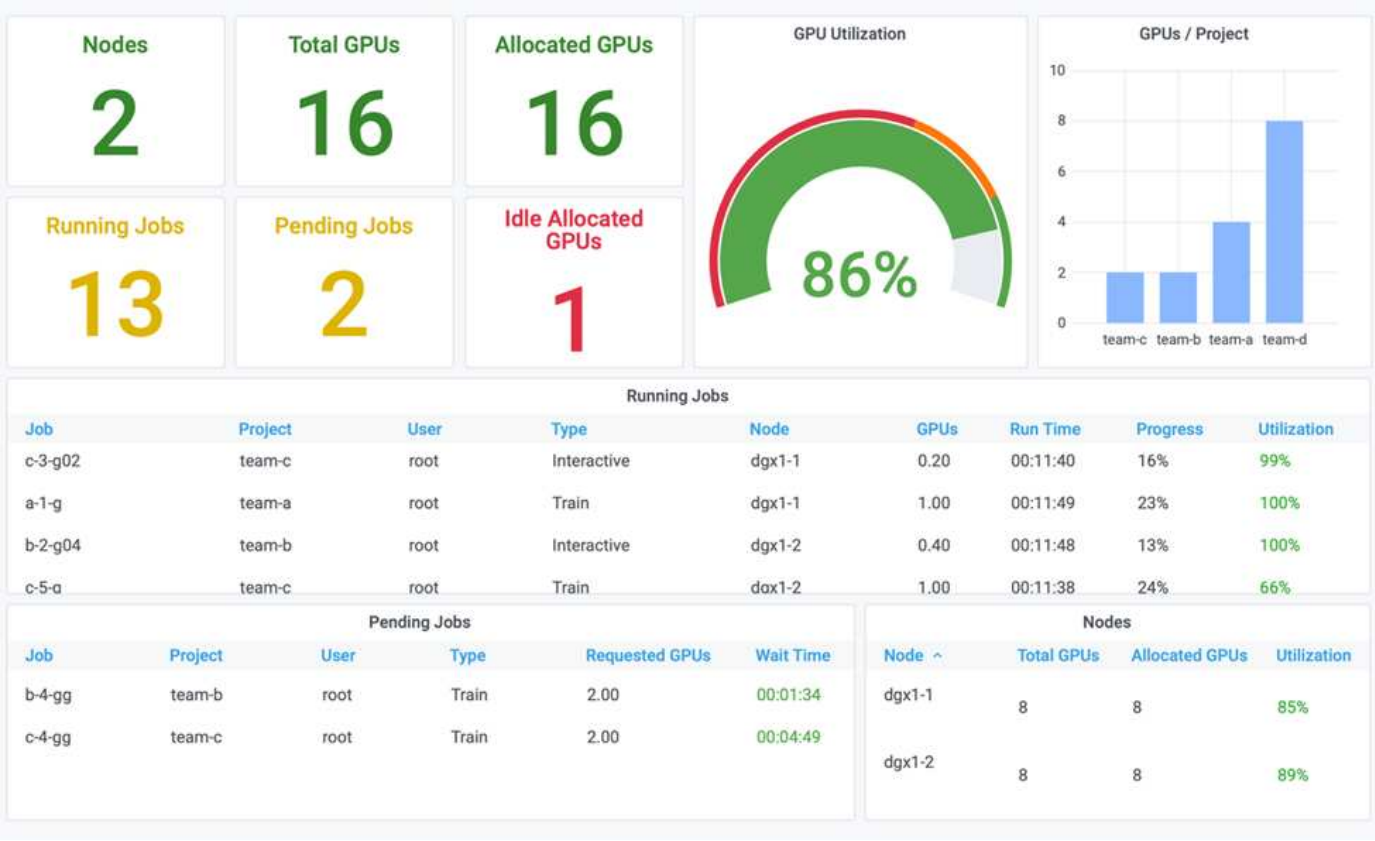
- Der Cluster sollte voll sein: Es werden 16/16 GPUs verwendet.
- Hohe Cluster-Auslastung.
- Mehr Experimente als GPUs aufgrund der fraktionalen Zuweisung:
- `team-d` Nutzt nicht die gesamte Quote; daher `team-b` Und `team-c` Da die Experimente mit zusätzlichen GPUs durchgeführt werden können, werden Innovationen schneller beschleunigt.

Gerechtigkeit Bei Der Grundlegenden Ressourcenzuweisung

In diesem Abschnitt zeigen wir das, wann `team-d` Fragt nach mehr GPUs (sie sind unter ihrem Kontingent), unterbricht das System die Workloads von `team-b` Und `team-c` Und bewegt sie in einen auslaufenden Staat auf faire Weise.

Details, einschließlich Stellenausschreibungen, verwendete Container-Images und ausgeführte Befehlssequenzen, finden Sie im Abschnitt "Testdetails für Abschnitt 4.9".

Die folgende Abbildung zeigt die resultierende Cluster-Auslastung, die pro Team zugewiesenen GPUs und ausstehende Aufgaben aufgrund von automatischem Lastausgleich und präventivem Scheduling. Wir können beobachten, dass, wenn die Gesamtzahl der GPUs, die von allen Team-Workloads angefordert werden, die Gesamtzahl der verfügbaren GPUs im Cluster übersteigt, Run:der interne Fairness-Algorithmus von AI für jeden einen Job unterbricht `team-b` Und `team-c` Weil sie ihre Projektquote erreicht haben. So lässt sich eine insgesamt hohe Cluster-Auslastung erzielen, während Data-Science-Teams nach wie vor unter von einem Administrator festgelegten Ressourcenbeschränkungen arbeiten.



Die Ergebnisse dieses Testszenarios zeigen Folgendes:

- **Automatischer Lastenausgleich.** das System gleicht die Quote der GPUs automatisch aus, so dass jedes Team jetzt seine Quote nutzt. Die Workloads, die angehalten wurden, gehören zu Teams, die ihr Kontingent überschreiten.
- **Fair Share Pause.** das System wählt, die Arbeitsbelastung eines Teams zu stoppen, das über seiner Quote war und dann die Arbeitsbelastung des anderen Teams zu stoppen. Run:AI hat interne Fairness-Algorithmen.

Gerechtigkeit Wegen Zu Viel Quoten

In diesem Abschnitt erweitern wir das Szenario, in dem mehrere Teams Workloads einreichen und ihre Kontingente übertreffen. Auf diese Weise zeigen wir, wie Run:AI Fairness-Algorithmus Clusterressourcen entsprechend dem Verhältnis voreingestellter Quoten zuweist.

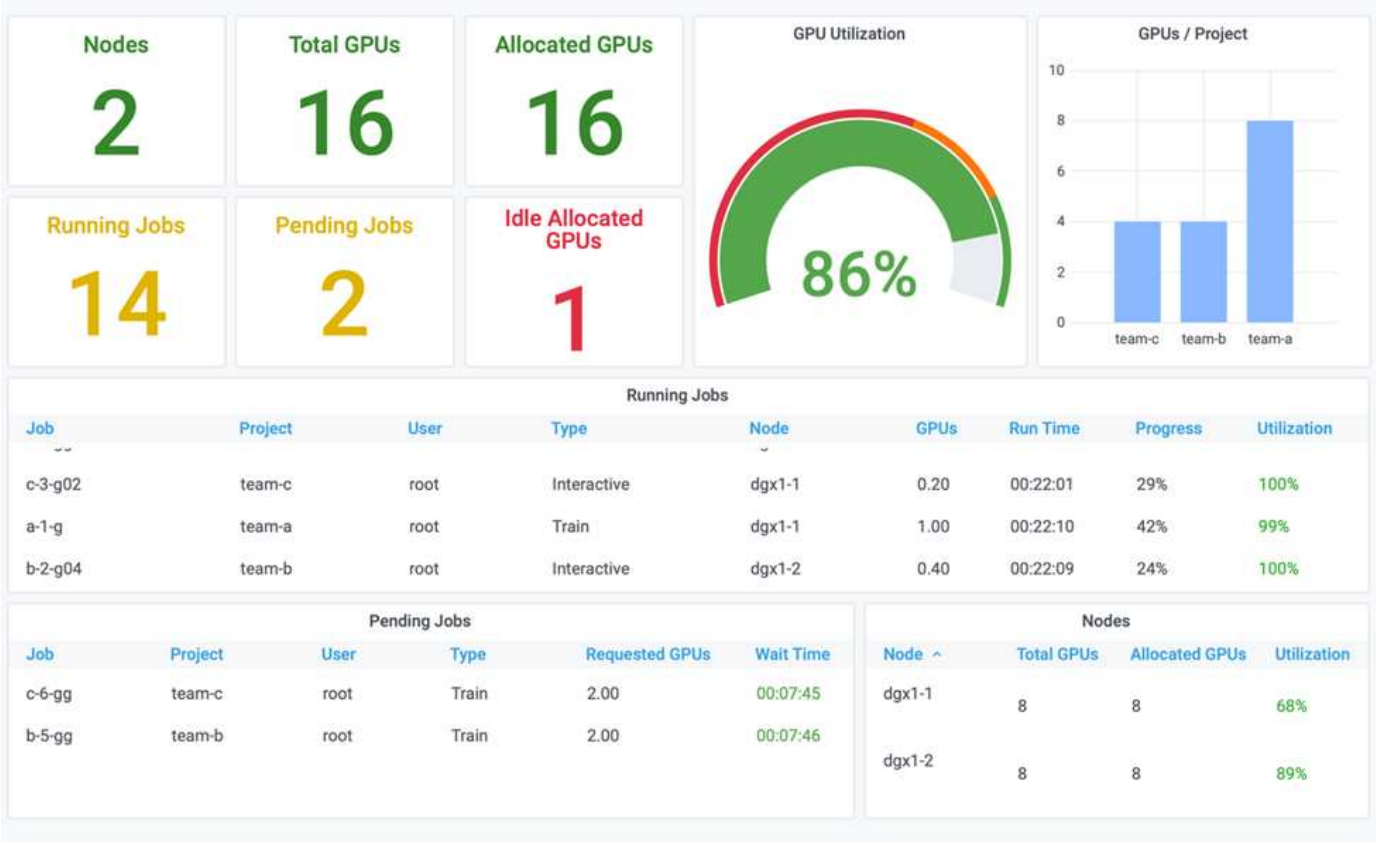
Ziele für dieses Testszenario:

- Warteschlangenmechanismus anzeigen, wenn mehrere Teams GPUs über ihre Kontingente anfordern
- Zeigen Sie, wie das System einen fairen Anteil des Clusters zwischen mehreren Teams verteilt, die entsprechend dem Verhältnis zwischen ihren Quoten über ihrem Kontingent liegen, so dass das Team mit dem höheren Kontingent einen größeren Anteil der freien Kapazität erhält.

Am Ende von "Gerechtigkeit Bei Der Grundlegenden Ressourcenzuweisung", Es gibt zwei Workloads in die Warteschlange: Einen für `team-b` Und eins für `team-c`. In diesem Abschnitt werden zusätzliche Workloads Warteschlange gestellt.

Details wie Stellenausschreibungen, verwendete Container-Images und ausgeführte Befehlssequenzen finden Sie unter "Testdetails für Abschnitt 4.10".

Wenn alle Jobs gemäß Abschnitt gesendet werden "Testdetails für Abschnitt 4.10", Das System-Dashboard zeigt das an `team-a`, `team-b`, und `team-c` Alle haben mehr GPUs als ihre voreingestellten Quoten. `team-a` Belegt vier mehr GPUs als die voreingestellte Soft Quota (vier), wohingegen `team-b` Und `team-c` Jeder besetzen zwei mehr GPUs als ihre Soft-Quota (zwei). Das Verhältnis der zugewiesenen GPUs zu viel Kontingent entspricht dem ihrer voreingestellten Quote. Das liegt daran, dass das System das voreingestellte Kontingent als Referenz der Priorität verwendet und entsprechend bereitgestellt hat, wenn mehrere Teams mehr GPUs anfordern und ihre Quoten übertreffen. Ein solcher automatischer Lastausgleich ist Fairness und Priorisierung, wenn Datenwissenschaftler im Unternehmen aktiv an der Entwicklung und Produktion von KI-Modellen arbeiten.



Die Ergebnisse dieses Testszenarios zeigen Folgendes:

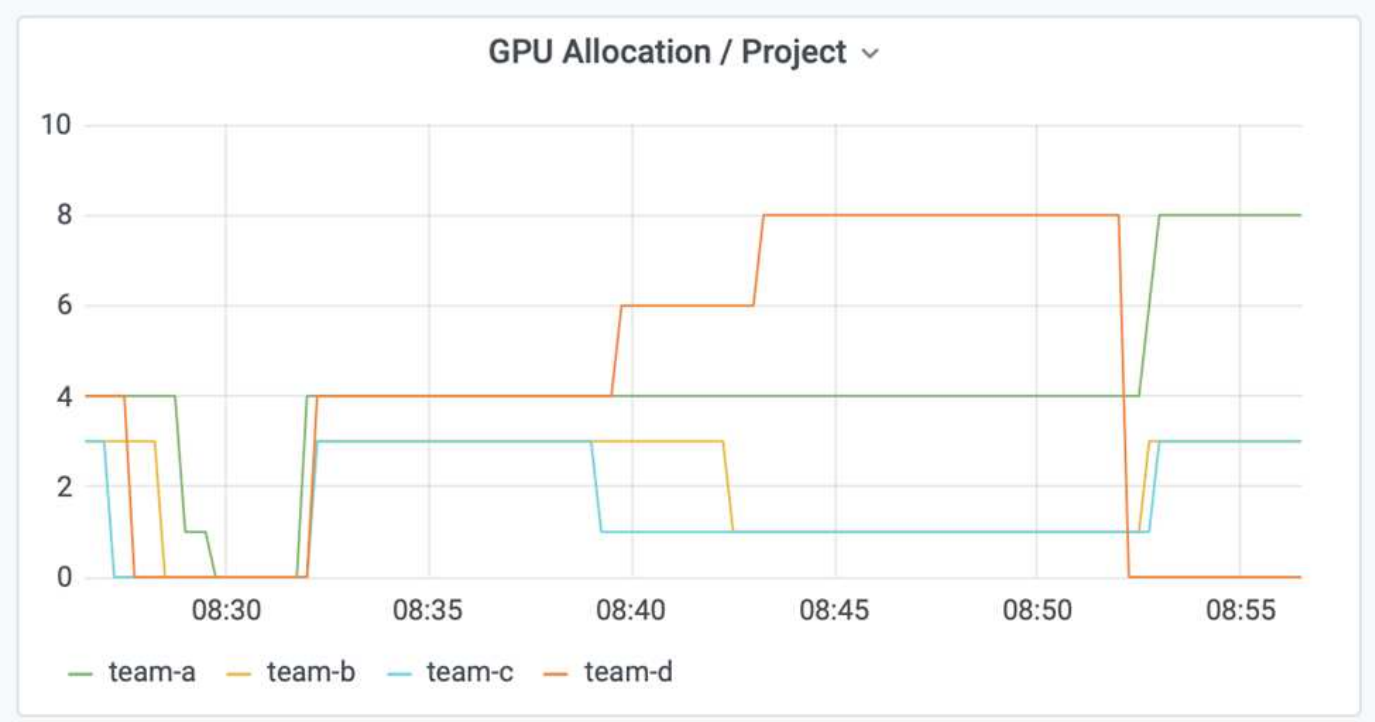
- Das System beginnt, die Workloads anderer Teams in die Warteschlange zu stellen.
- Die Reihenfolge der Abwarteschlangen wird nach Fairness-Algorithmen festgelegt, so dass `team-b` Und `team-c` Erhalten Sie die gleiche Menge von Over-Quota GPUs (da sie eine ähnliche Quote haben), und `team-a` Erhält eine doppelte Menge von GPUs, da ihre Quote doppelt so hoch ist wie die Quote von `team-b` Und `team-c`.
- Die gesamte Zuweisung erfolgt automatisch.

Daher sollte sich das System in folgenden Staaten stabilisieren:

Projekt	GPUs zugewiesen	Kommentar
Team A	8/4	Vier GPUs über die Quote. Leere Warteschlange.
Team b	4/2	Zwei GPUs über dem Kontingent. Ein Workload in Warteschlange.
Team c	4/2	Zwei GPUs über dem Kontingent. Ein Workload in Warteschlange.
Team d	0/8	GPU verwendet überhaupt nicht; keine Workloads in der Warteschlange.

Die folgende Abbildung zeigt die GPU-Zuweisung pro Projekt im Dashboard „Run:AI Analytics“ für die Abschnitte "Erreichen einer hohen Cluster-Auslastung mit GPU-Zuweisung über ein Kontingent", "Gerechtigkeit Bei Der Grundlegenden Ressourcenzuweisung", und "Gerechtigkeit Wegen Zu Viel Quoten".

Die einzelnen Zeilen in der Abbildung geben die Anzahl der GPUs an, die jederzeit für ein Data-Science-Team bereitgestellt werden. Wie wir sehen, weist das System GPUs dynamisch entsprechend den eingereichten Workloads zu. So können Teams im Cluster GPUs zur Verfügung stehen und Jobs entsprechend der Fairness vorbeugen, bevor sie endlich einen stabilen Zustand für alle vier Teams erreichen.



Speichern von Daten in einem mit Trident bereitgestellten PersistenzVolume

NetApp Trident ist ein vollständig unterstütztes Open-Source-Projekt, damit Sie die anspruchsvollen Persistenzanforderungen Ihrer Container-Applikationen erfüllen können. Daten können in ein mit Trident bereitgestelltes Kubernetes PersistentVolume (PV) gelesen und geschrieben werden. Hinzu kommen Daten-Tiering, Verschlüsselung, NetApp Snapshot Technologie, Compliance und hohe Performance der Datenmanagement-Software NetApp ONTAP.

Verwendung von PVCs in einem vorhandenen Namespace

Bei größeren KI-Projekten könnte es effizienter sein, dass unterschiedliche Container Daten in dasselbe Kubernetes PV lesen und schreiben. Zur Wiederverwendung einer Kubernetes Persistent Volume Claim (PVC) muss der Anwender bereits eine PVC erstellt haben. Siehe "NetApp Trident Dokumentation" Für Details zur Erstellung eines PVC. Hier ein Beispiel für die Wiederverwendung einer vorhandenen PVC:

```
$ runai submit pvc-test -p team-a --pvc test:/tmp/pvc1mount -i gcr.io/run-ai-demo/quickstart -g 1
```

Führen Sie den folgenden Befehl aus, um den Status des Jobs anzuzeigen `pvc-test` Für Projekt `team-a`:

```
$ runai get pvc-test -p team-a
```

Sie sollten die PV /tmp/pvc1Mount auf sehen `team-a` Job `pvc-test`. Auf diese Weise können mehrere Container aus demselben Volume gelesen werden. Das ist hilfreich, wenn es mehrere konkurrierende Modelle in der Entwicklung oder in der Produktion gibt. Data Scientists können ein Modellensemble aufbauen und dann Vorhersageergebnisse durch Mehrheitsvotum oder andere Techniken kombinieren.

Verwenden Sie Folgendes für den Zugriff auf die Container-Shell:

```
$ runai bash pvc-test -p team-a
```

Anschließend können Sie das gemountete Volume prüfen und auf Ihre Daten innerhalb des Containers zugreifen.

Diese Funktion zur Wiederverwendung von VES funktioniert mit NetApp FlexVol Volumes und NetApp ONTAP FlexGroup Volumes. Damit können Data Engineers flexiblere und robustere Datenmanagement-Optionen nutzen, um die NetApp Data Fabric nutzen zu können.