مقیاس بندی خودکار
سرویس دهی مدل‌های زبانی بزرگ

LLaMA by Meta

پروپوزال

۲

محمد سعید صفایی صادق

مفاهیم پایه

Large Language Models (LLMs)

Training آموزش یعنی چی؟

Training یعنی مدل رو با استفاده از حجم زیادی داده، آموزش بدیم تا یاد بگیره چطور کار کنه.

در این مرحله:

مدل از صفر یا از قبل‌آموزش‌دیده pretrained شروع می‌کنه

بهش متن‌های زیاد داده می‌شه

مدل سعی می‌کنه توکن بعدی رو حدس بزنه

اگر اشتباه کنه، از طریق backpropagation وزن‌هاش تنظیم می‌شن

این کار بارها تکرار می‌شه تا مدل یاد بگیره

Inference استنتاج یا اجرا یعنی چی؟

Inference یعنی از یک مدل آموزش‌دیده استفاده کنیم تا پیش‌بینی انجام بده یا جواب بده.

در این مرحله:

وزن‌های مدل ثابت هستن

مدل هیچ چیزی یاد نمی‌گیره

فقط ورودی می‌گیره و خروجی می‌ده

Inference = استفاده از مدل برای پاسخ‌دهی

مقایسه Training vs Inference

ویژگی	(آموزش) Training	(استنتاج) Inference
هدف	یاد گرفتن	استفاده از آموخته‌ها
وزن‌ها	تغییر می‌کنن	ثابت هستن
هزینه محاسباتی	بسیار سنگین	معمولاً سبک‌تر
داده‌ها	زیاد و متنوع	کم (مثلاً فقط یک سوال)
سرعت	خیلی کندتر	خیلی سریع‌تر
مثال	آموزش LLaMA روی داده	چت‌کردن با LLaMA

LLaMA

LLaMA مخفف Large Language Model Meta AI است.

این یک سری از مدل‌های زبانی بزرگ LLMs است که توسط شرکت Meta فیسبوک سابق توسعه داده شده‌اند و برای کارایی بالا، **منابع کمتر** و امکان استفاده پژوهشی/صنعتی طراحی شده‌اند.

برخلاف مدل‌هایی مانند GPT-3 که میلیون‌ها دلار سخت‌افزار نیاز دارند، LLaMA برای این طراحی شده که:

- روی منابع محدودتر (مثلاً چند GPU قابل‌دسترس عمومی) قابل اجرا باشد.
- به پژوهشگران و توسعه‌دهندگان مستقل امکان دهد مدل‌های قدرتمند را تست و fine-tune کنند.
- در حوزه پژوهش متن‌باز مشارکت کند.

مفاهیم

Fine-tuning

آموزش دادن دوباره یک مدل از پیش‌آموزش‌دیده مثل LLaMA یا GPT روی یک مجموعه داده جدید و خاص تا مدل برای کار خاصی بهتر عمل کنه.

به زبان ساده، یه مدل بزرگ قبلاً «زبان رو یاد گرفته»، حالا فقط داریم بهش می‌گیم "حالا اینو برای این کار دقیق‌تر یاد بگیر."

مدل‌های بزرگ مثل LLaMA یا GPT روی میلیاردها کلمه از اینترنت آموزش دیدن، ولی:

به طور کلی آموزش دیدن general purpose ، نه برای کار خاص مثل ترجمه حقوقی یا چت‌بات پزشکی.

شاید در زمینه خاص (مثلاً متن‌های فارسی یا متون علمی) دقت کافی نداشته باشن.

مفاهیم

Fine-tuning

فرض کن LLaMA رو داریم. می‌خوایم:

یه چت‌بات پزشکی بسازیم.

مدل باید دقیق به پرسش‌های پزشکی پاسخ بده.

کاری که می‌کنیم:

داده‌هایی از دیالوگ دکتر- بیمار یا مقاله‌های پزشکی آماده می‌کنیم.

LLaMA رو روی این داده‌ها fine-tune می‌کنیم.

خروجی: یک LLaMAکه رفتار و اطلاعاتش پزشکی‌تره.

روش‌های مختلف Fine-tuning

1-Full fine-tuning:

همه وزن‌های مدل به‌روزرسانی می‌شن (دقیق ولی سنگین).

2-LoRA (Low-Rank Adaptation):

فقط چند لایه جدید آموزش می‌دیم ← سریع‌تر، کم‌هزینه‌تر.

3-Prompt tuning / Adapter tuning:

بدون تغییر مدل اصلی، فقط ورودی‌ها یا میان‌لایه‌ها رو دستکاری می‌کنیم.

Hugging Face چیست؟

Hugging Face یک شرکت و پلتفرم متن‌بازه که ابزارها و منابعی برای پردازش زبان طبیعی NLP، بینایی ماشین CVو یادگیری ماشین ML فراهم می‌کنه.

مهم‌ترین محصولش:

کتابخانه‌ی معروف Transformers که اجرای مدل‌هایی مثل BERT، GPT، T5، LLaMAو... رو فوق‌العاده ساده کرده.

Hugging Face

KVCache چیست؟

KVCache مخفف Key-Value Cache است.

این یک ساختار حافظه‌ای است که در مدل‌های ترنسفورمر Transformer برای ذخیره‌سازی نتایج میانی در حین تولید توکن‌ها (کلمات) استفاده می‌شود.

از نظر فنی در ترنسفورمر، در هر لایه‌ی Attention، دو ماتریس مهم تولید می‌شن:

K (Keys)

V (Values)

برای هر توکن جدید، فقط ماتریس Q (Query) جدید ساخته می‌شه و با K/Vقبلی (که Cacheشدن) محاسبه می‌شه.

کاربرد اصلی KVCache

در مدل‌هایی مثل LLaMA یا GPT که به‌صورت auto-regressive متن تولید می‌کنند، هر توکن باید بر اساس تمام توکن‌های قبلی ساخته شود.

بدون KVCache :

مدل برای تولید هر توکن، باید تمام توکن‌های قبلی را دوباره از صفر پردازش کند ← خیلی کند و پرهزینه است.

با KVCache :

مدل نتایج میانی Keys و Values از مراحل قبلی را در حافظه نگه می‌دارد، تا فقط توکن جدید را پردازش کند ← سریع و بهینه.

کاربرد اصلی KVCache

در چت‌بات‌ها:

هنگام تولید جواب طولانی، بدون KVCache زمان تولید هر کلمه طولانی‌تر و کندتر می‌شه

با KVCache، حتی با ورودی بلند، تولید سریع باقی می‌مونه.

KVCache یعنی حافظه‌ای برای نگهداری نتایج Attention قبلی، تا مدل لازم نباشه توکن‌های قبلی رو دوباره پردازش کنه.

این کلید افزایش سرعت inference در مدل‌هایی مثل GPT، LLaMAو PaLM هست.

Auto-regressive decoding یعنی چی؟

مدل به‌صورت گام‌به‌گام (توکن‌به‌توکن)، متن تولید می‌کنه.

هر توکن جدید براساس توکن‌های قبلی تولید می‌شه.

مدل همیشه از خروجی‌های قبلی خودش استفاده می‌کنه تا توکن بعدی رو پیش‌بینی کنه.

فرض کن مدل می‌خواد جمله‌ی «سلام دنیا!» رو بنویسه.

فرآیند به صورت زیره:

مدل ورودی: "سلام" ← خروجی: "دنیا"

حالا ورودی می‌شه: "سلام دنیا" ← خروجی: "!"

حالا ورودی می‌شه: "سلام دنیا !" ← پایان (توکن < <eosیا پایان جمله)

یعنی توکن دوم بر اساس اولی، سوم بر اساس اول و دوم و الی آخر ساخته می‌شه.

Auto-regressive decoding یعنی چی؟

چطور کار می‌کنه در Transformer ؟

مدل در مرحله training یاد می‌گیره که هر توکن رو از روی توکن‌های قبلی تولید کنه (با loss به‌نام causal loss)

۱- مدل یک توکن تولید می‌کنه

۲- اون توکن به دنباله اضافه می‌شه

۳- مدل با دنباله‌ی جدید دوباره توکن بعدی رو تولید می‌کنه

۴- تا رسیدن به طول دلخواه یا توکن پایان eos

چون گام‌به‌گام اجرا می‌شه، نسبت به روش‌های non-autoregressive که همه توکن‌ها رو با هم تولید می‌کنن) کندتره

به همین دلیله که مفاهیمی مثل **KVCache** یا **speculative decoding** ابداع شدن تا سرعتش بالا بره.

Auto-regressive decoding یعنی چی؟

ویژگی	توضیح
روش تولید متن	گام‌به‌گام، توکن‌به‌توکن
وابستگی	هر توکن به توکن‌های قبلی
مزایا	دقیق، قابل‌کنترل
معایب	نسبتاً کند
مدل‌هایی که استفاده می‌کنن	GPT, LLaMA, PaLM, Falcon و همه مدل‌های auto-regressive decoder-only

Prefill و Decode یعنی چی؟

در مدل‌های زبانی auto-regressive مثل LLaMA، فرآیند تولید متن به دو بخش تقسیم می‌شه:

هر بار که یک مدل زبانی مثل LLaMAیا GPTشروع به تولید متن می‌کنه، اول می‌ره سراغ Prefill، و بعد وارد Decode می‌شه.

مرحله ۱ Prefill

همیشه اول، مدل جمله‌ی تو رو کامل می‌خونه

مرحله ۲ Decode

بعد از Prefill، حالا مدل می‌خواد جواب تولید کنه

[ورودی کاربر] → 🧠 Prefill → ✍️ Decode → [پاسخ مدل]

Prefill و Decode یعنی چی؟

۱. Prefill پیش‌پرکردن:

زمانی که مدل، ورودی اولیه prompt رو می‌گیره و برای اولین بار، همه توکن‌هاش رو پردازش می‌کنه.

در این مرحله:

تمام توکن‌های prompt از ابتدا تا انتها، به‌صورت یکجا پردازش می‌شن.

مدل Key و Value های مربوط به همه این توکن‌ها رو می‌سازه و در KVCache ذخیره می‌کنه.

محاسبات این بخش معمولاً سنگین‌تره چون تمام لایه‌ها و توکن‌ها با هم پردازش می‌شن.

Prefill و Decode یعنی چی؟

۲. Decode رمزگشایی / تولید مرحله‌به‌مرحله

بعد از prefill، مدل شروع به تولید توکن‌های جدید یک‌به‌یک می‌کنه، تا مثلاً به انتهای جمله برسه.

در این مرحله:

هر بار فقط یک توکن جدید تولید می‌شه.

توکن جدید به prompt قبلی اضافه می‌شه.

فقط Query محاسبه می‌شه، و با KVCache قبلی ترکیب می‌شه.

به لطف KVCache، decode خیلی سریع‌تر از prefill هست.

Prefill و Decode یعنی چی؟

چرا این تقسیم‌بندی مهمه؟

برای بهینه‌سازی زمان پاسخ latency در inference

برای Disaggregation تفکیک بین سرورهای مختلف ← مثلاً prefill و decode روی سرورهای جداگانه انجام بشه

برای تعیین معیارهای عملکرد مدل:

زمان prefill = TTFT (Time to First Token)

زمان decode = TBT (Time Between Tokens)

Token generation یعنی چی؟

توکن Token یعنی یک تکه از متن که می‌تونه یک کلمه، بخش از کلمه، یا حتی علامت باشه.

Token generation یعنی مدل مثل LLaMA یا GPT، توکن‌ها (کلمات یا بخش‌هایی از کلمات) رو یکی‌یکی تولید می‌کنه.

"سلام دنیا!" → [سلام, دنیا, !]

Time-to-First-Token (TTFT) ؟

مدت‌زمانی که طول می‌کشه اولین توکن تولید بشه.

یعنی از وقتی ورودی رو به مدل می‌دی، تا وقتی اولین کلمه از جواب ساخته بشه

شامل مرحله‌ی Prefill می‌شه، چون مدل باید اول ورودی رو کامل پردازش کنه.

مثال: کاربر می‌پرسه: «سلام! خوبی؟»

مدل بعد از ۶۰۰ میلی‌ثانیه می‌گه: «من...» ⟵

$$\text{TTFT} = 600\text{ms}$$

نکته: در TTFT معمولاً یک مرحله‌ی Decode هم انجام می‌شود برای تولید اولین توکن.

Time-Between-Tokens (TBT) ؟

فاصله‌ی زمانی بین تولید توکن اول تا توکن دوم، دوم تا سوم و ...

یعنی زمان بین هر دوتا کلمه‌ای که مدل تولید می‌کنه.

"من | خوبم | مرسی"

TBT1 = 200ms بین "من" و "خوبم"

TBT2 = 220ms بین "خوبم" و "مرسی"

شاخص	اهمیت
TTFT	نشون می‌ده کاربر **چقدر صبر می‌کنه تا جواب شروع شه**
TBT	نشون می‌ده مدل **با چه سرعتی ادامه‌ی جواب رو تولید می‌کنه**

FlashInfer چیست؟

FlashInfer یک کتابخانه‌ی inference سریع high-performance برای اجرای مدل‌های زبانی بزرگ مثل LLaMA، GPTو غیره روی GPU است که توسط تیم‌هایی مثل vLLM و Colossal-AI توسعه پیدا کرده.

هدفش اینه که decode تولید توکن رو تا حد ممکن سریع و بهینه انجام بده.

بیان ساده:

موتور سریع اجرای decode روی GPU

FlashInfer

FlashInfer چه کاری انجام میدهد؟

۱- بهینه‌سازی عملیات decode مرحله‌ی تولید توکن

یعنی TBT را کم میکند.

۲- استفاده از تکنیک‌های CUDA پیشرفته برای سرعت بیشتر

۳- اجرای چند درخواست به‌صورت batch موازی multi-query batching

یعنی درخواست های زیاد همزمان اجرا می شود

۴- بهینه‌سازی memory access برای استفاده بهتر از GPU cache / VRAM **GPU**

یعنی استفاده از gpu بهینه می شود

Tensor Parallelism یعنی چی؟

Tensor Parallelism یعنی تقسیم محاسبات لایه‌های مدل (مثلاً لایه‌های ترنسفورمر) بین چند GPU ، به‌جای اینکه کل مدل روی یک GPU باشه.

مثلاً:

مدل یک لایه Attention داره

به‌جای اینکه کامل روی یک GPU باشه

ماتریس‌های بزرگ اون لایه بین چند GPU تقسیم می‌شن

و هر GPU فقط بخشی از اون محاسبات رو انجام می‌ده

Tensor Parallelism یعنی چی؟

فرض کن لایه‌ی attention یه ماتریس ۴۰۹۶×۴۰۹۶ داره.

با Tensor Parallelism روی ۴ GPU :

GPU0 محاسبه‌ی ستون‌های ۰–۱۰۲۳

GPU1 ستون‌های ۱۰۲۴–۲۰۴۷

GPU2 ستون‌های ۲۰۴۸–۳۰۷۱

GPU3 ستون‌های ۳۰۷۲–۴۰۹۵

و در نهایت خروجی‌ها جمع می‌شن **reduce** .

مزایا:

کاهش نیاز به حافظه در هر GPU

مناسب برای inference و training مدل‌های بزرگ

افزایش سرعت در مدل‌های بزرگ

مفاهیم پایه
Autoscaling / Model Serving

Model Autoscaling

Model autoscaling یعنی سیستم به‌طور خودکار تصمیم می‌گیره چند نسخه instance از یک مدل باید فعال باشه، تا بتونه با افزایش یا کاهش تعداد درخواست‌ها هماهنگ بشه.

یعنی چی؟

وقتی تعداد درخواست‌ها زیاد می‌شه، سیستم:

- **مدل رو روی GPU های بیشتری اجرا می‌کنه scale up**

و وقتی درخواست‌ها کم می‌شن:

- **بعضی مدل‌ها رو خاموش می‌کنه تا منابع آزاد بشن scale down**

Model Autoscaling

مدل‌های زبانی بزرگ مثل LLaMA، GPT خیلی سنگین هستن:

اجرای دائمی اون‌ها روی چند GPU خیلی گرونه

ولی گاهی درخواست‌ها خیلی کمه و نیازی نیست همه مدل‌ها فعال باشن

پس:

۱- autoscaling باعث می‌شه فقط وقتی نیاز هست، منابع مصرف بشن

۲- هم هزینه کمتر بشه

۳- هم کاربران با تأخیر latency کم پاسخ بگیرن

Model Autoscaling

Autoscaling چطور کار می‌کنه؟

۱- سیستم ترافیک ورودی رو مانیتور می‌کنه

۲- اگر تعداد درخواست‌ها زیاد شه ← instance جدید از مدل اجرا می‌شه

۳- اگر درخواست کم شه ← مدل از روی بعضی GPUها برداشته می‌شه

۴- گاهی از caching یا KVCache برای سرعت بیشتر استفاده می‌شه

Serverless Model-as-a-Service (MAAS)

Model-as-a-Service (MAAS) یعنی یک سیستم یا پلتفرم که به کاربران اجازه می‌ده مدل‌های یادگیری ماشین مثل LLaMA یا BERT رو آپلود کنن و استفاده کنن، بدون اینکه درگیر زیرساخت بشن.

و وقتی می‌گیم Serverless MAAS، یعنی:

این سرویس به‌صورت خودکار، پویا و بدون نیاز به مدیریت مستقیم سرورها مدل‌ها رو اجرا، مقیاس‌بندی و خاموش/روشن می‌کنه.

Model-as-a-Service (MAAS)

فرض کن می‌خوای یه مدل چت‌بات داشته باشی:

با MAAS فقط مدل‌تو آپلود می‌کنی و یک API بهت می‌ده که باهاش صحبت کنی، خودش پشت صحنه GPUها، حافظه، autoscaling و غیره رو مدیریت می‌کنه

با serverless MAAS :

- حتی نیاز نداری مدل همیشه روشن باشه
- فقط وقتی درخواست بیاد، مدل فعال می‌شه
- بعد از مدتی بی‌استفاده بودن، مدل خاموش می‌شه
- همه‌چی خودکار و مقیاس‌پذیر انجام می‌شه

Model-as-a-Service (MAAS)

ویژگی‌های کلیدی Serverless MAAS

ویژگی	توضیح
بدون سرور (serverless)	لازم نیست GPU یا سرور رو خودت رزرو یا مدیریت کنی
مقیاس‌پذیری خودکار	وقتی ترافیک زیاد شد، خودش مدل رو روی GPUهای بیشتر اجرا می‌کنه
پرداخت به ازای مصرف	فقط برای زمانی که از مدل استفاده می‌کنی هزینه می‌دی
پشتیبانی از چند مدل	همزمان می‌تونی ده‌ها یا صدها مدل رو آپلود و مدیریت کنی
پشتیبانی از Cold Start	مدل‌ها فقط وقتی لازم باشن فعال می‌شن (البته با تأخیر اولیه)

Model-as-a-Service (MAAS)

نمونه های واقعی

نام سرویس / پروژه	توضیح کوتاه	وضعیت استفاده
Hugging Face Inference API	اجرای مدل‌ها (مثل GPT، BERT، LLaMA) بدون نیاز به زیرساخت؛ پرداخت به‌ازای مصرف	✅ تجاری، عمومی
vLLM + FastAPI	اجرای سریع LLMها با پشتیبانی از batching و KVCache؛ مناسب برای MAAS خودساخته	✅ متن‌باز، قابل‌پیاده‌سازی
ServerlessLLM	سیستم تخصصی برای cold start handling، autoscaling و serving مدل‌های LLM	🔬 پژوهشی / مقاله‌ای
Modal.com	پلتفرم ابری برای اجرای مدل‌های ML/LLM با ساختار serverless؛ تمرکز روی simplicity	✅ تجاری، نیاز به ثبت‌نام
Replicate.com	اجرای مدل‌های متن‌باز (مثل Stable Diffusion، LLaMA، Whisper) به‌صورت API	✅ تجاری، پرداخت به‌ازای استفاده
BlitzServe / BlitzScale	سیستم serverless پیشرفته برای inference سریع، معرفی‌شده در مقاله مربوط به Blitz	🔬 پژوهشی / نیازمند GPU و تنظیمات پیشرفته

Serving Instance یعنی چی؟

یک نمونه‌ی فعال از مدل مثل LLaMA یا GPT که در حال اجرا روی یک GPU یا CPU هست و آماده‌ست تا به درخواست‌های کاربر پاسخ بده.

یعنی هر **"کپی زنده از مدل"** که در یک لحظه داره کار می‌کنه، یک serving instance محسوب می‌شه.

وقتی کاربران زیادی هم‌زمان درخواست می‌فرستن، یا مدل رو باید روی چند GPU تقسیم کنیم مثلاً با tensor parallelism یا batching :

۱ درخواست ← ۱ instance کافیه

۱۰۰ درخواست در ثانیه ← ۱۰ یا ۲۰ instance همزمان نیاز داریم

Serving Instance یعنی یک مدل زبانی که همین الان روی سخت‌افزار GPU یا CPU لود شده، در حال اجراست، و آماده‌ست به درخواست‌ها پاسخ بده.

Goodput یعنی چی؟

Goodput یعنی تعداد توکن‌های مفید و واقعی که مدل در واحد زمان برای کاربر تولید می‌کنه.

به عبارت دیگه:

فقط توکن‌هایی که واقعاً به دست کاربر می‌رسن و معنا دارن (نه محاسبات داخلی یا هدر رفته)

واحدش معمولاً هست: tokens per second (toks/sec)

مفهوم	تعریف	شامل چی می‌شه؟
Throughput	کل توکن‌هایی که مدل تولید کرده (حتی اگه دور ریخته شدن یا به کاربر نرسیدن)	ممکنه شامل تأخیر یا بلااستفاده هم باشه
✅ Goodput	فقط توکن‌هایی که واقعاً به کاربر ارسال شدن و استفاده شدن	دقیق‌تر و واقعی‌تر از نگاه کاربر

Goodput یعنی چی؟

مثال واقعی

فرض کن یه مدل LLaMA روزی ۱۰،۰۰۰ توکن تولید کنه:

۲،۰۰۰ تا در صف افتادن و هرگز به کاربر نرسیدن

۸،۰۰۰ تا واقعاً به کاربر برگشتن

در این حالت:

Throughput = 10,000 toks/day

Goodput = 8,000 toks/day

Goodput یعنی چی؟

چرا Goodput مهمه؟

برای ارزیابی درست کیفیت و کارایی سیستم LLM، باید بدونیم:

آیا فقط داریم سریع محاسبه می‌کنیم؟ (Throughputبالا)

یا واقعاً به‌موقع، مفید و کاربرمحور جواب می‌دیم؟ (Goodputبالا)

هدف :

Goodput را تا جای ممکن نزدیک به Throughput کنیم.

Goodput

چه عواملی Goodput را کاهش می‌دن؟

عامل	تأثیر
صف‌های طولانی (queueing delay)	توکن‌ها دیر می‌رسن یا اصلاً نمی‌رسن
بار زیاد روی GPU	سرعت پاسخ‌دهی کم می‌شه
cold start / cache miss	زمان انتظار بالا → توکن‌ها دیر تولید می‌شن
بازیابی منابع یا توقف‌ها (stop-the-world loading)	تأخیر شدید برای اولین توکن (TTFT بالا)

SLO (Service Level Objective)

SLO یعنی «هدف سطح سرویس» – یه قرارداد عددی یا هدف مشخص که سیستم باید برای کاربر رعایت کنه.

این هدف نشون می‌ده که:

سرویس باید چقدر سریع، دقیق، در دسترس یا قابل اعتماد باشه

عملکرد سرویس باید چطور اندازه‌گیری و تضمین بشه

واژه	تعریف	مثال
SLI (Service Level Indicator)	عدد واقعی اندازه‌گیری شده	TTFT = 420ms
SLO	هدفی که برای SLI مشخص کردی	٪95 برای TTFT < 500ms
SLA (Service Level Agreement)	قرارداد رسمی با مشتری	اگر SLO رعایت نشد → جریمه یا غرامت

SLO (Service Level Objective)

چرا SLO در LLM serving مهم است؟

۱- برای مدیریت quality of service (QoS)

۲- برای فعال‌کردن auto-scaling دقیق

۳- برای تصمیم‌گیری در مورد لود بالانسینگ، caching، prioritization

۴- برای حفظ تجربه خوب کاربر نهایی

Disaggregation یعنی چی؟

Disaggregation یعنی «تفکیک کردن بخش‌های مختلف یک سیستم» که قبلاً با هم بودن، تا بتونیم هر بخش رو جدا مدیریت و مقیاس‌دهی کنیم.

در دنیای LLM Serving، منظور از Disaggregation چیه؟

به‌طور خاص:

P/D Disaggregation مخفف:

P = Prefill

D = Decode

مرحله	وظیفه
Prefill (P)	پردازش اولیه ورودی (prompt)، ساختن KVCache
Decode (D)	تولید توکن‌ها، یکی‌یکی، با استفاده از KVCache

Disaggregation یعنی چی؟

پس Disaggregation چه کمکی می‌کنه؟

۱- می‌تونی سرورهایی برای prefill داشته باشی (GPUهای بزرگ، حافظه زیاد)

۲- و جدا از اون، سرورهایی برای decode کوچکتر، سریع‌تر

۳- هر بخش رو جدا مقیاس‌دهی autoscalingکنی

۴- بهتر بتونی burst workloadرو کنترل کنی

۵- باعث کاهش تأخیر کلی TTFT + TBT می‌شه

Stop-the-world loading یعنی چی؟

Stop-the-world loading یعنی زمانی که سیستم می‌خواد یک مدل LLM مثل LLaMA رو بارگذاری load کنه، همه چیز باید متوقف بشه و صبر کنه تا بارگذاری کامل بشه.

یعنی:

- هیچ توکنی تولید نمی‌شه
- هیچ درخواستی پاسخ داده نمی‌شه
- تا وقتی که کل وزن‌های مدل از دیسک (یا شبکه) به حافظه GPU منتقل بشن

Stop-the-world loading

چه زمانی اتفاق می‌افته؟

در حالت‌های مثل:

Cold Start: وقتی مدلی اصلاً توی حافظه نیست و تازه باید load بشه

Autoscaling: وقتی می‌خوای مدل رو روی یک GPU جدید اجرا کنی

Model Swapping: اگر چند مدل مختلف داری و یکی از حافظه بیرون رفته باشه

Stop-the-world loading

فرض کن کاربر درخواست جدیدی می‌فرسته و مدل مربوطه هنوز توی GPU نیست:

- سیستم باید اول وزن‌های مدل رو از دیسک یا storage بارگذاری کنه مثلاً از S3
- این کار ممکنه ۱۰ ثانیه طول بکشه
- در این مدت هیچ جوابی داده نمی‌شه
- وقتی لود تموم شد، اولین توکن تولید می‌شه
- TTFT = 10 ثانیه یا بیشتر

Stop-the-world loading

راه‌حل‌های مقابله با Stop-the-world loading

راه‌حل	توضیح
Preloading / Warm Pools	چند مدل همیشه آماده باشن (مثل "گرم نگه‌داشتن")
Progressive loading	بخشی از مدل سریع بارگذاری بشه، decode با تأخیر شروع شه
Disaggregation (P/D)	فقط بخش prefill دچار بارگذاری بشه، decode unaffected
Speculative Execution	پاسخ موقتی از مدل مشابه یا نسخه کوچک‌تر تا اصلی آماده شه
Async Loading	درخواست‌ها صف بشن و سیستم بارگذاری رو همزمان با مدیریت صف انجام بده

Burst Workload یعنی چی؟

Burst workload یعنی زمانی که به‌طور ناگهانی و موقتی تعداد زیادی درخواست request به سیستم وارد می‌شن، معمولاً خیلی بیشتر از حد نرمال.

یعنی:

- یه موج بزرگ از کاربران، همزمان درخواست می‌دن
- سیستم باید بتونه بهشون پاسخ بده، بدون اینکه هنگ کنه یا کند بشه

Burst Workload یعنی چی؟

Burst workload یعنی زمانی که به‌طور ناگهانی و موقتی تعداد زیادی درخواست request به سیستم وارد می‌شن، معمولاً خیلی بیشتر از حد نرمال.

یعنی:

- یه موج بزرگ از کاربران، همزمان درخواست می‌دن
- سیستم باید بتونه بهشون پاسخ بده، بدون اینکه هنگ کنه یا کند بشه

مشکل Burst Workload

چالش	توضیح
TTFT بالا	مدل هنوز scale نشده، بار زیاده، صف تشکیل می‌شه
Cold start	مدل باید بارگذاری بشه (stop-the-world loading رخ می‌ده)
Timeout یا Drop	درخواست‌ها رد می‌شن چون ظرفیت سیستم پره
SLO failure	تأخیر بالا، تجربه بد کاربر، نقض SLA/SLO

سیستم خوب چطور با burst کنار میاید

راهکار	کاربرد
Autoscaling سریع (Live autoscaling)	مدل‌ها سریع بالا بیان
Prefetch / Prewarm	چند instance از قبل آماده باشن
Disaggregation (P/D)	prefill و decode جدا scale بشن
Queuing & Throttling	درخواست‌ها صف‌بندی یا محدود بشن
Load shedding	در صورت overload، بعضی درخواست‌ها رد شن ولی سیستم نخوابه

Scaling Abstraction یعنی چی؟

Scaling Abstraction یعنی ایجاد یک لایه انتزاعی برای مدیریت مقیاس‌پذیری سیستم، به طوری که کاربر یا توسعه‌دهنده مجبور نباشه جزئیات پیچیده مقیاس‌پذیری (مثل توزیع بار، تخصیص منابع یا مدیریت سرورها) رو مدیریت کنه.

برای مثال، فرض کن یه مدل LLaMA روی یک سرور داری.

وقتی درخواست‌ها زیاد می‌شه، سیستم می‌تونه مقیاس‌پذیری scaling رو فعال کنه، مثلاً به صورت خودکار منابع بیشتری اضافه کنه یا درخواست‌ها رو بین چند سرور تقسیم کنه.

این اتفاق از دید کاربر کاملاً پنهان می‌مونه و کاربر فقط مدل و API رو استفاده می‌کنه.

Scaling Abstraction یعنی چی؟

Scaling Abstraction یک مفهوم طراحی است که هدف آن مدیریت مقیاس‌پذیری به‌طور خودکار در سیستم‌های مختلف است.

این مفهوم در ابزارها و پلتفرم‌های مختلف مثل Kubernetes، Serverless Platforms، یا Cloud Functions پیاده‌سازی شده و به‌طور مستقیم کمک می‌کند تا توسعه‌دهندگان نیازی به نگرانی در مورد مقیاس‌بندی منابع نداشته باشند.

بنابراین، Scaling Abstraction خودش یک مفهوم است که توسط ابزارها و پلتفرم‌های مختلف به صورت یک ویژگی یا سرویس پیاده‌سازی می‌شود.

ابزارهایی که از Scaling Abstraction استفاده می‌کنند

۱- **Kubernetes**: یک پلتفرم مدیریت کانتینر که به‌طور خودکار مقیاس‌دهی و توزیع بار را انجام می‌دهد.

۲- **AWS Lambda**: سرویس Serverless که مقیاس‌دهی خودکار توابع را بر اساس نیازهای ورودی انجام می‌دهد.

۳- **Google App Engine**: یک پلتفرم PaaS که به‌طور خودکار برنامه‌ها را مقیاس‌دهی می‌کند.

۴- **Azure Functions**: سرویس FaaS که به‌طور خودکار توابع را مقیاس می‌دهد.

۵- **Docker Swarm**: ابزاری برای مدیریت خوشه‌های Docker که مقیاس‌دهی و توزیع بار را بین کانتینرها انجام می‌دهد.

۶- **Cloud Foundry**: پلتفرم PaaS که به‌طور خودکار مقیاس‌دهی و مدیریت برنامه‌ها را انجام می‌دهد.

کانتینر چیست؟

کانتینر یک محیط اجرایی ایزوله است که برای اجرای برنامه‌ها و سرویس‌ها طراحی شده است.

در واقع، کانتینرها برنامه‌ها و تمام وابستگی‌های لازم برای اجرای آن‌ها (مانند کتابخانه‌ها، تنظیمات، فایل‌ها و غیره) را در خود بسته‌بندی می‌کنند.

ویژگی‌های کلیدی کانتینر:

۱- **ایزولاسیون**: کانتینرها به‌طور ایزوله از همدیگر اجرا می‌شوند، به این معنا که هیچ‌کدام از کانتینرها به منابع یا تنظیمات کانتینر دیگر دسترسی ندارند.

۲- **پورتابل بودن**: کانتینرها می‌توانند روی هر دستگاه یا محیطی که از سیستم‌های کانتینری مانند Docker پشتیبانی می‌کند، اجرا شوند.

۳- **سبک بودن**: کانتینرها از سیستم‌عامل میزبان استفاده می‌کنند، بنابراین نسبت به ماشین‌های مجازی سبک‌تر و سریع‌تر هستند.

۴- **مقیاس‌پذیری**: کانتینرها می‌توانند به‌راحتی مقیاس داده شوند، یعنی می‌توان تعداد زیادی کانتینر را برای تحمل بارهای بیشتر به‌طور خودکار اجرا کرد.

تشبیه کانتینر به pdf چیست؟

مشابه PDF که در هر سیستم قابل مشاهده است، یک کانتینر حاوی کد و وابستگی‌ها است که در هر محیطی که از آن پشتیبانی کند اجرا می‌شود، بدون اینکه نیاز به نصب وابستگی‌های اضافی باشد.

مثلاً یک برنامه‌ای که در یک کانتینر Docker بسته‌بندی شده باشد، می‌تواند روی هر سیستمی که از Docker پشتیبانی می‌کند اجرا شود، چه در ویندوز، مک، یا لینوکس.

بدون کانتینر مثلاً PHP :

فرض کنید می‌خواهید یک برنامه PHP را اجرا کنید.

در این صورت، شما باید PHP و دیگر ابزارهای مورد نیاز را به طور مستقل روی سیستم خود نصب کنید. اگر نسخه‌های مختلف PHP یا وابستگی‌های مختلف نیاز باشد، باید به‌طور دستی نصب و پیکربندی کنید.

کانتینر

با کانتینر مثلاً Docker :

در اینجا، شما یک کانتینر برای برنامه خود می‌سازید که همه چیز (PHP، کتابخانه‌ها، پیکربندی‌ها، و سایر وابستگی‌ها) را در داخل خود دارد.

این کانتینر به‌طور ایزوله اجرا می‌شود و وقتی آن را در سیستم خود اجرا کنید، به هیچ‌گونه نصب اضافی نیاز ندارید.

سیستم شما به سادگی از کانتینر استفاده می‌کند که همه‌چیز در داخل آن بسته‌بندی شده است.

وقتی از کانتینرها استفاده می‌کنید، به‌جای نصب و پیکربندی نرم‌افزارها به‌طور دستی، می‌توانید تمام وابستگی‌ها را داخل کانتینر قرار داده و آن را در هر محیطی که از کانتینر پشتیبانی کند اجرا کنید، دقیقاً مثل PDF که در هر سیستمی بدون نیاز به نصب نرم‌افزار خاصی قابل باز شدن است.

Kubernetes

با استفاده از ابزاری مانند Kubernetes، شما می‌توانید کانتینرها را به‌طور خودکار توزیع و مقیاس‌پذیر روی سیستم‌های مختلف (نودها) اجرا کنید.

Kubernetes این کار را به‌طور هوشمند و خودکار انجام می‌دهد و شما نیازی به مدیریت دستی توزیع بار یا مقیاس‌دهی ندارید.

Kubernetes پیچیدگی‌های مقیاس‌پذیری را برای شما پنهان می‌کند و فقط شما نیاز به تعریف و مدیریت سرویس‌ها و برنامه‌ها دارید. این ابزار به‌طور خودکار منابع را تخصیص می‌دهد، بار را بین سرورها توزیع می‌کند و در صورت نیاز به مقیاس‌دهی به‌طور خودکار این کار را انجام می‌دهد.

مفاهیم

نحوه عملکرد **Kubernetes** در مقیاس‌پذیری و توزیع کانتینرها:

۱- ایجاد Pod ها:

در Kubernetes ، شما برنامه‌های خود را در قالب Pod قرار می‌دهید.

هر Pod ممکن است شامل یک یا چند کانتینر باشد.

Kubernetes این Pods را بین نودهای مختلف خوشه توزیع می‌کند.

مفاهیم

نحوه عملکرد **Kubernetes** در مقیاس‌پذیری و توزیع کانتینرها:

۲- کنترل مقیاس با ReplicaSet و Deployment :

- اگر بخواهید تعداد مشخصی از Pods در هر زمان اجرا شوند، از ReplicaSet استفاده می‌کنید.

به‌عنوان مثال، اگر می‌خواهید ۳ کانتینر از یک برنامه همیشه در حال اجرا باشند، Kubernetes این را مدیریت می‌کند.

- Deployment به شما این امکان را می‌دهد که مقیاس‌دهی را خودکار کنید و از Rolling Updates استفاده کنید.

نحوه عملکرد **Kubernetes** در مقیاس‌پذیری و توزیع کانتینرها:

۳- افزایش یا کاهش مقیاس به‌طور خودکار:

- Horizontal Pod Autoscaler (HPA) در Kubernetes به‌طور خودکار تعداد Pods را بر اساس بار کاری و کارایی سیستم (مانند CPU یا حافظه) تنظیم می‌کند.
- زمانی که ترافیک زیاد می‌شود، Kubernetes به‌طور خودکار Pods جدید ایجاد می‌کند.

نحوه عملکرد Kubernetes در مقیاس‌پذیری و توزیع کانتینرها:

1- Vertical Scaling (Mening Resources per Pod)

Vertical Scaling یا مقیاس‌دهی عمودی به معنی افزایش منابع مثل حافظه RAM و پردازنده CPU برای یک Pod خاص هست.

در مقیاس‌دهی عمودی، تعداد Pods تغییر نمی‌کنه، بلکه به هر Pod اختصاص داده‌شده به‌طور مستقیم منابع بیشتری مثل CPU و RAM تخصیص داده می‌شود.

به‌عنوان مثال، اگر یک Pod نیاز به منابع بیشتری برای پردازش ترافیک داشته باشد، شما می‌توانید منابع آن را افزایش دهید (مثلاً به آن CPU بیشتر یا حافظه بیشتر بدهید).

نحوه عملکرد Kubernetes در مقیاس‌پذیری و توزیع کانتینرها:

2- Horizontal Pod Autoscaler (HPA) :

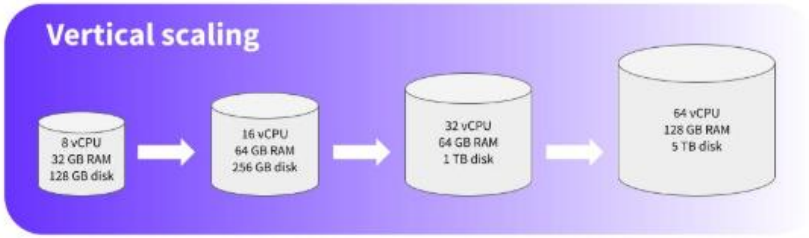
Horizontal Pod Autoscaler به شما این امکان رو می‌ده که تعداد Pods کانتینرها رو افزایش یا کاهش بدید بر اساس بار کاری مثل استفاده از CPU یا حافظه.

HPA تعداد Pods که هر Pod ممکنه شامل یک یا چند کانتینر باشه رو برای یک سرویس یا برنامه تنظیم می‌کنه.

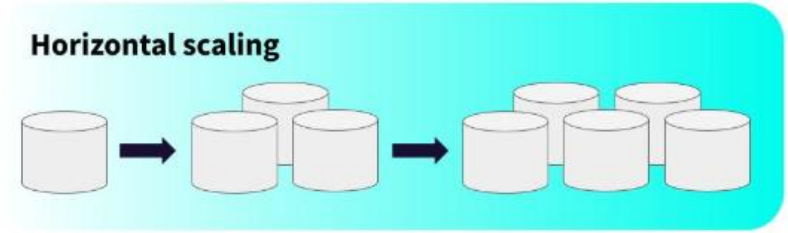
به عبارت ساده‌تر، وقتی بار ترافیکی بالا میره، HPA به‌طور خودکار تعداد Pods رو افزایش می‌ده تا ظرفیت بیشتری برای پردازش درخواست‌ها داشته باشید. وقتی بار کم می‌شه، تعداد Pods رو به‌طور خودکار کاهش می‌ده.

یادآوری مقیاس بندی عمودی و افقی:

مقیاس‌دهی عمودی: فرض کنید یک وب‌سایت دارید که روی یک سرور با ۴ هسته پردازنده و ۱۶ گیگابایت رم اجرا می‌شود. اگر ترافیک وب‌سایت افزایش یابد، شما ممکن است تصمیم بگیرید که **این سرور را ارتقا دهید** به ۸ هسته پردازنده و ۳۲ گیگابایت رم تا بتوانید بار بیشتری را پردازش کنید.



مقیاس‌دهی افقی: همان وب‌سایت را در نظر بگیرید. به جای ارتقای سرور، شما **یک سرور جدید به سیستم اضافه می‌کنید** تا ترافیک را بین سرورها توزیع کنید. حالا شما ۲ یا چند سرور دارید که به‌طور همزمان درخواست‌ها را پردازش می‌کنند.



مفاهیم

Layer-level scaling

یا مقیاس‌پذیری در سطح لایه یک مفهوم در معماری سیستم‌های توزیع‌شده است که به مقیاس‌دهی و مدیریت منابع در لایه‌های مختلف یک سیستم اشاره دارد.

چیستی Layer-level Scaling :

در سیستم‌های پیچیده، معمولاً یک برنامه یا سرویس به چندین لایه تقسیم می‌شود. هر لایه ممکن است مسئول یک بخش خاص از کار باشد، مثلاً:

لایه فرانت‌اند (رابط کاربری)

لایه بک‌اند (منطق برنامه)

لایه دیتابیس (مدیریت داده‌ها)

مقیاس‌پذیری در سطح لایه یعنی شما می‌توانید به‌طور مستقل هر لایه از سیستم را مقیاس‌دهی کنید تا منابع مورد نیاز برای هر بخش به‌طور بهینه و جداگانه مدیریت شود.

Layer-level scaling

فرض کنید تعداد کاربران شما افزایش یافته و نیاز دارید که مقیاس‌دهی انجام دهید:

۱- شما می‌توانید **فقط لایه فرانت‌اند را مقیاس دهید** و سرورهای بیشتری برای پاسخ‌دهی به **درخواست‌های کاربران** اضافه کنید.

۲- اگر **پردازش‌های پیچیده بک‌اند کندتر شده باشند**، تنها لایه بک‌اند را مقیاس داده و **سرورهای بیشتری برای پردازش** درخواست‌ها اضافه کنید.

۳- اگر **داده‌ها رشد کرده** و درخواست‌های دیتابیس زیاد شده‌اند، تنها لایه دیتابیس را مقیاس دهید، **مثلاً با توزیع داده‌ها بین چند پایگاه داده.**

مفاهیم

Zigzag scheduling

در Zigzag scheduling ، کارها یا درخواست‌ها به‌طور مناسب و متناوب بین منابع مختلف (مثلاً سرورها یا خوشه‌ها) توزیع می‌شود.

هدف این الگوریتم پیش‌بینی و بهینه‌سازی توزیع منابع است که در آن بار یا ترافیک به‌طور منظم و متناوب بین منابع موجود تقسیم می‌شود.

- در این الگوریتم، زمانی که منابع (مثل سرورها یا پردازنده‌ها) به کارهای مختلف تخصیص داده می‌شوند، این تخصیص به‌طور زیکزاک انجام می‌شود.

- این به این معناست که کارها به صورت منظم و با الگویی متناوب توزیع می‌شوند که به نوعی خود تنظیم است، به‌طوری که هیچ‌کدام از منابع درگیر یا تحت فشار زیاد قرار نمی‌گیرند.

Zigzag scheduling

تصور کنید یک سیستم توزیع‌شده با چندین سرور دارید که باید به تعدادی درخواست پاسخ دهد. در این سیستم، درخواست‌ها به‌طور زیکزاک توزیع می‌شوند:

سرور ۱ یک درخواست را پردازش می‌کند.

سرور ۲ درخواست بعدی را پردازش می‌کند.

سرور ۳ به درخواست بعدی اختصاص داده می‌شود.

سپس دوباره سرور ۱ درخواست بعدی را پردازش می‌کند، و این روند به‌طور متناوب ادامه می‌یابد.

این نوع توزیع منابع باعث می‌شود که هیچ سروری تحت فشار زیادی قرار نگیرد و کارها به‌طور منظم بین منابع تقسیم شوند.

مفاهیم

Live Autoscaling:

Live autoscaling به فرآیندی اشاره دارد که به‌طور خودکار و در زمان واقعی live تعداد منابع (مانند سرورها یا کانتینرها) را بر اساس نیاز سیستم افزایش یا کاهش می‌دهد.

این کار بدون وقفه در عملکرد سیستم انجام می‌شود و به‌طور خودکار و داینامیک منابع را به اندازه نیاز مقیاس می‌دهد.

فرض کنید یک وب‌سایت دارید که در زمان‌های خاصی مانند بلک فرایدی ترافیک زیادی دارد.

سیستم به‌طور خودکار در زمان واقعی تعداد سرورهای خود را برای پردازش تعداد بیشتر درخواست‌ها افزایش می‌دهد.

بعد از پایان ترافیک زیاد، تعداد سرورها به‌طور خودکار کاهش پیدا می‌کند تا منابع بهینه مصرف شوند.

مفاهیم

Cooperative Execution:

Cooperative execution به مفهوم همکاری منابع یا واحدهای اجرایی (مثل سرورها یا کانتینرها) در انجام کارها و توزیع منابع به‌طور بهینه اشاره دارد.

در این روش، منابع به‌طور مشارکتی و هم‌زمان با هم کار می‌کنند تا بار کاری به شکل بهینه تقسیم شود.

فرض کنید یک سیستم پردازش داده بزرگ دارید.

در این سیستم، پردازش‌های مختلف می‌توانند به‌طور هم‌زمان در سرورهای مختلف اجرا شوند و با همکاری یکدیگر، کارهایی که به هم وابسته هستند را پردازش کنند.

این مدل باعث می‌شود که زمان پردازش کاهش پیدا کند و عملکرد بهینه حفظ شود.

Live Autoscaling vs Cooperative Execution:

ویژگی	Live Autoscaling	Cooperative Execution
تعریف	مقیاس‌دهی منابع به‌طور خودکار و در زمان واقعی	همکاری منابع برای پردازش همزمان و بهینه
هدف	افزایش یا کاهش منابع بر اساس نیاز	استفاده بهینه و هماهنگ از منابع
کاربرد	**پردازش بار کاری متغیر** (مثل وب‌سایت‌ها)	**همکاری بین منابع** برای کاهش زمان پردازش
مزیت	مقیاس‌دهی در زمان واقعی بدون وقفه	بهبود عملکرد و کاهش زمان تأخیر

Live Autoscaling به‌طور خودکار تعداد منابع (سرورها یا کانتینرها) را بر اساس تغییرات در زمان واقعی مقیاس می‌دهد.

Cooperative execution به معنای هماهنگی و همکاری منابع است تا کارها به‌طور بهینه پردازش شوند و از منابع به بهترین شکل استفاده شود.

مفاهیم

Resource Wastage یا هدر رفت منابع

در Cooperative Execution ، به‌ویژه در صورتی که منابع به‌طور هماهنگ و هم‌زمان با هم کار کنند، ممکن است هدررفت منابع resource wastage بیشتر شود.

این مشکل ممکن است در مواردی به وجود بیاید که منابع در حال همکاری هستند اما به طور بهینه از آن‌ها استفاده نمی‌شود.

در Cooperative Execution، چندین منبع (مانند سرور یا کانتینر) ممکن است به‌طور همزمان برای پردازش یک وظیفه یا بار کاری کار کنند.

اما اگر این منابع به‌طور هماهنگ و بدون تقسیم بهینه کارها کار کنند، ممکن است برخی از منابع به طور کم‌کار underutilized باقی بمانند و در نتیجه منجر به هدررفت منابع شوند.

مفاهیم

Parameter Loading

Parameter loading فرآیندی است که در آن پارامترهای مدل (مانند وزن‌ها، بایاس‌ها، و تنظیمات دیگر) از داده‌های ذخیره‌شده در دیسک یا سرور به حافظه اجرایی (معمولاً RAMیا VRAMدر کارت گرافیک) بارگذاری می‌شوند تا برای پردازش و استنتاج در مدل‌های یادگیری ماشین یا برنامه‌های پیچیده استفاده شوند.

۱- **بارگذاری پارامترها**: ابتدا پارامترهای مدل از داده‌های ذخیره‌شده (مثل دیسک سخت یا فضای ذخیره‌سازی ابری) به حافظه اصلی RAM بارگذاری می‌شوند.

۲- **استفاده در پردازش**: پس از بارگذاری، مدل می‌تواند این پارامترها را در فرآیند استنتاج یا آموزش استفاده کند.

۳- **توزیع و پردازش**: در صورت نیاز، برای بهینه‌سازی سرعت پردازش، بارگذاری پارامترها به GPU یا واحدهای پردازش ویژه مثل TPU می‌تواند انجام شود.

مفاهیم

Parameter Loading

نکته:

Cold Start به‌طور خاص به بارگذاری پارامترها و منابع از فضای ذخیره‌سازی به حافظه اشاره دارد که به دلیل حجم داده‌ها و زمان مورد نیاز برای انتقال، باعث تاخیر می‌شود.

Parameter Loading در این فرآیند (یعنی تاخیر شروع سرد) نقش اساسی دارد و می‌تواند باعث زمان تأخیر بالا شود، به‌ویژه در سیستم‌های بزرگ و پیچیده.

برای کاهش این تاخیر، از روش‌هایی مثل استفاده از حافظه سریع‌تر یا پیش‌بارگذاری می‌توان استفاده کرد.

مفاهیم

Scaling Stall Time

Scaling Stall Time زمان تاخیر یا محدودیت زمانی است که هنگام مقیاس‌دهی سیستم (مثلاً افزودن منابع جدید یا انتقال منابع بین سرورها) به‌وجود می‌آید. این ممکن است زمانی رخ دهد که:

۱- سیستم به کندی منابع جدید اضافه می‌کند یا منابع قدیمی را آزاد می‌کند.

۲- زمان انتقال داده‌ها یا منابع از یک نود به نود دیگر به طور قابل توجهی طول می‌کشد.

۳- توزیع یا هماهنگی بار در سیستم پیچیده است و باعث توقف یا کندی می‌شود.

Scaling Stall Time زمان توقف یا تأخیری است که هنگام مقیاس‌دهی سیستم به دلیل تخصیص منابع، انتقال داده‌ها، یا هماهنگی پیچیده بین منابع به‌وجود می‌آید.

این تأخیر ممکن است باعث کاهش عملکرد و تجربه کاربری منفی شود.

مفاهیم پایه
Caching / Data Plane

مفاهیم

Cache Miss:

Cache Miss زمانی اتفاق می‌افتد که داده‌ای که درخواست شده، در کش (حافظه سریع) موجود نباشد و باید از منبع اصلی (مانند دیسک یا سرور داده) بارگذاری شود.

این باعث می‌شود که زمان پاسخ‌دهی بیشتر شود، زیرا به جای اینکه داده‌ها از حافظه سریع کش گرفته شوند، باید از منابع اصلی سیستم دریافت شوند که معمولاً کندتر است.

فرض کنید یک سرویس وب در حال پردازش داده‌ها است.

برای افزایش سرعت پردازش، سیستم کشی برای ذخیره‌سازی نتایج پیشین طراحی شده است.

اگر کاربر درخواست داده‌ای کند که قبلاً در کش ذخیره نشده باشد، سیستم باید داده‌ها را از پایگاه داده اصلی بارگذاری کند، که به‌طور معمول کندتر از کش است و باعث می‌شود Cache Miss رخ دهد.

مفاهیم

Cache Hit:

Cache Hit زمانی رخ می‌دهد که داده‌های مورد نظر در کش (حافظه سریع) موجود باشند و سیستم قادر باشد تا داده‌ها را بلافاصله از کش بدون نیاز به دسترسی به منابع کندتر مانند دیسک یا پایگاه داده بازیابی کند.

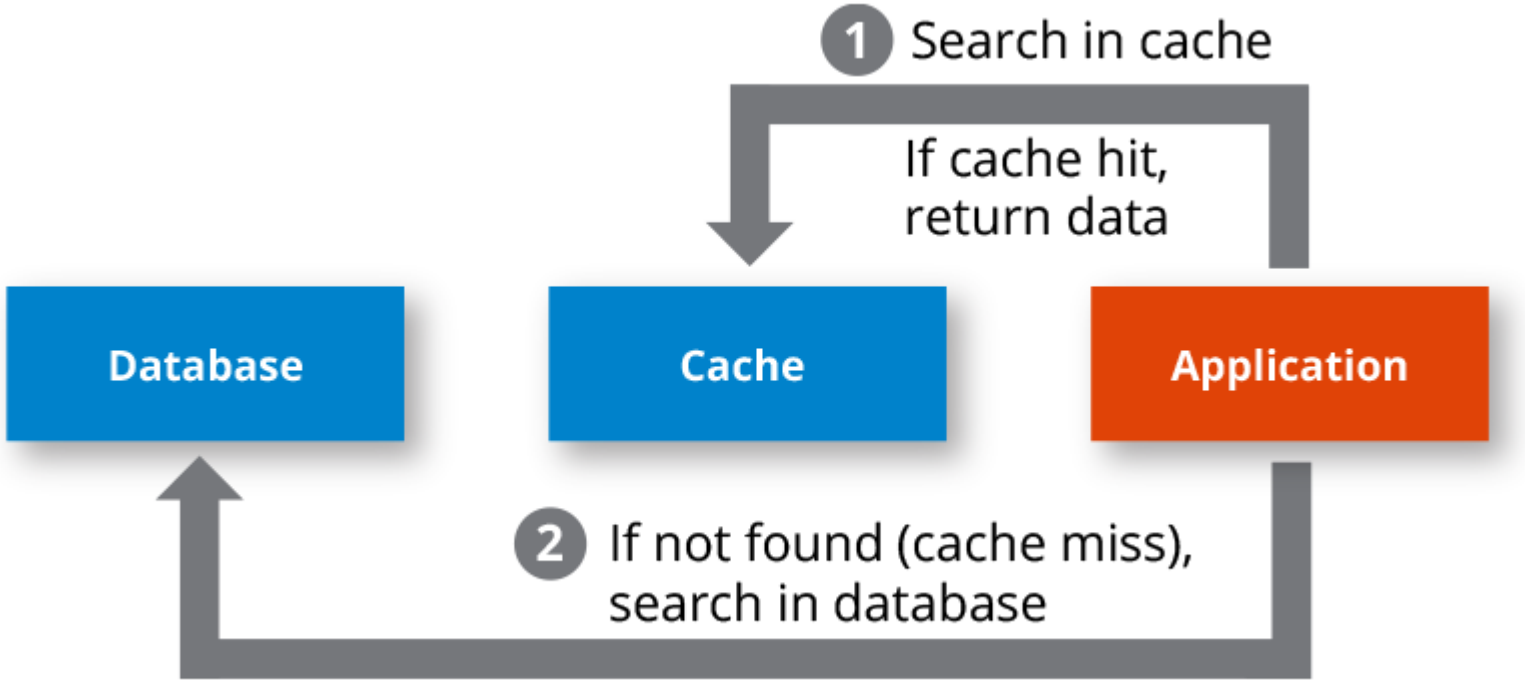
این فرآیند باعث کاهش زمان تأخیر و افزایش کارایی سیستم می‌شود.

۱- **داده‌ها در کش ذخیره می‌شوند**: وقتی که سیستم یا برنامه برای اولین بار داده‌ای را بارگذاری می‌کند (مثلاً از پایگاه داده)، این داده‌ها معمولاً در کش ذخیره می‌شوند تا در درخواست‌های بعدی سریع‌تر در دسترس قرار گیرند.

۲- **درخواست جدید به کش ارسال می‌شود**: وقتی یک درخواست مشابه برای همان داده‌ها دوباره وارد می‌شود، سیستم ابتدا به کش سر می‌زند.

۳- **داده در کش موجود است**: اگر داده‌ها در کش موجود باشند، سیستم آن‌ها را بلافاصله از کش بازیابی می‌کند که به این مرحله Cache Hit می‌گوییم.

Cache Miss and Cache Hit:



مفاهیم

Host Cache:

Host Cache به کشی اشاره دارد که در سیستم‌های محلی یا سرورها (نودها) ذخیره می‌شود.

این کش می‌تواند شامل داده‌هایی باشد که در حافظه سرور یا میزبان قرار دارند و می‌توانند به‌طور سریع در دسترس قرار گیرند.

Host Cache برای کاهش زمان دسترسی به داده‌ها و کاهش فشار روی منابع اصلی (مثل پایگاه داده یا سیستم‌های ذخیره‌سازی) طراحی شده است.

مفاهیم

Remote Cache:

داده‌ها در سیستم‌های دیگر ذخیره می‌شوند: برخلاف Host Cache که در حافظه محلی سرور ذخیره می‌شود، داده‌ها در کش‌های دور (مثلاً کش‌های توزیع‌شده یا کش‌های مبتنی بر شبکه مانند Memcached یا Redis ذخیره می‌شوند.

دسترسی به داده‌ها از طریق شبکه: این کش‌ها معمولاً از طریق شبکه قابل دسترسی هستند، بنابراین ممکن است زمان تأخیر بیشتری داشته باشند.

مقیاس‌پذیری بالا: کش‌های دور می‌توانند به‌طور مقیاس‌پذیر به چندین سرور یا نود توزیع شوند، که باعث می‌شود بار بیشتری را تحمل کنند.

مفاهیم

O(1) Chaching:

O(1) Caching به الگوریتم‌های کشی اطلاق می‌شود که دسترسی به داده‌ها در کش را در زمان ثابت انجام می‌دهند.

این بدین معناست که زمان دسترسی به داده‌ها مستقل از اندازه کش است و نیازی به جستجو در داده‌ها یا پردازش‌های پیچیده نداریم.

O(1) در اصطلاحات تحلیل الگوریتم‌ها به معنی زمان ثابت است، که به این معناست که عملیات دسترسی به داده‌ها هیچ‌گونه وابستگی به اندازه مجموعه داده‌ها ندارد و همیشه زمان ثابت می‌برد.

در Host Cache، که معمولاً به کش محلی سرور اشاره دارد (مانند حافظه RAM یا کش‌هایی که در یک سرور خاص ذخیره می‌شوند)، O(1) به راحتی قابل دستیابی است.

مفاهیم

O(1) Chaching:

O(1) دسترسی به داده‌ها در **کش‌های توزیع‌شده** نیز ممکن است، زیرا بسیاری از سیستم‌های کش توزیع‌شده از ساختارهای داده‌ای مشابه مانند Hash Map یا LRU Cache استفاده می‌کنند که می‌توانند به‌طور مؤثر و با زمان ثابت به داده‌ها دسترسی داشته باشند.

حتی در **کش‌های شبکه‌ای Network Cache** که داده‌ها بین چندین سرور یا سیستم توزیع می‌شوند، از ساختارهای داده‌ای سریع برای دسترسی سریع به داده‌ها استفاده می‌شود.

اگر الگوریتم کش به‌درستی پیاده‌سازی شود، O(1) دسترسی به داده‌ها در این کش‌ها هم امکان‌پذیر است.

Global Parameter Pool

در سیستم‌های پیچیده و توزیع‌شده، هنگامی که مدل‌های یادگیری ماشین یا محاسبات موازی اجرا می‌شوند، به‌طور معمول پارامترها (مثل وزن‌ها در شبکه‌های عصبی) **باید بین تمام نودها به اشتراک گذاشته شوند.**

در یادگیری ماشین توزیع‌شده: مدل‌های یادگیری ماشین ممکن است از پارامترهایی استفاده کنند که باید به‌طور مشترک توسط همه نودها یا سرورها در فرآیند آموزش استفاده شوند. در این حالت، یک Global Parameter Poolتمامی این پارامترها را در خود ذخیره می‌کند.

در سیستم‌های توزیع‌شده: پارامترهای پیکربندی مشترک در سیستم‌هایی که دارای چندین نود هستند (مانند Kubernetes یا سیستم‌های پردازش ابری) نیز می‌توانند در یک Global Parameter Pool نگهداری شوند **تا اطمینان حاصل شود که تمامی نودها از یک مجموعه یکسان از تنظیمات استفاده می‌کنند.**

مفاهیم

Global Parameter Pool

مثلا:

در آموزش مدل‌های Deep Learning، هنگامی که آموزش مدل در چندین سرور یا GPU انجام می‌شود، وزن‌ها و بایاس‌ها باید به‌طور یکسان و متمرکز در تمام سرورها به اشتراک گذاشته شوند.

یک Global Parameter Pool می‌تواند این وزن‌ها را ذخیره کرده و به تمام سرورها اجازه دهد تا در فرآیند آموزش از داده‌های یکسان استفاده کنند.

Multi-tier Caching

در Multi-tier caching، داده‌ها در چندین سطح مختلف کش ذخیره می‌شوند، که هر سطح می‌تواند ویژگی‌های متفاوتی از نظر سرعت، ظرفیت و هزینه‌های ذخیره‌سازی داشته باشد. این سطوح کش ممکن است شامل:

۱- **کش لایه اول L1 Cache**: کش سریع و کوچک، معمولاً در حافظه محلی (مثل RAM یا حافظه کش CPU) قرار دارد.

۲- **کش لایه دوم L2 Cache**: کش بزرگتر و کندتر نسبت به L1، اما همچنان سریعتر از دسترسی به منابع اصلی.

۳- **کش لایه سوم L3 Cache**: کش بزرگتر و معمولاً مشترک بین چند پردازنده که برای داده‌هایی که در L1 یا L2 موجود نیستند، استفاده می‌شود.

۴- **کش توزیع‌شده Distributed Cache**: در سیستم‌های مقیاس‌پذیر، داده‌ها ممکن است در کش‌های توزیع‌شده مانند Redis یا Memcached ذخیره شوند که داده‌ها را در چندین سرور یا نود پخش می‌کند.

Multi-tier Caching

داده‌ها به طور بهینه توزیع می‌شوند: درخواست‌ها ابتدا به کش لایه اول (مثلاً کش حافظه RAM) مراجعه می‌کنند.

اگر داده‌ها در این کش نباشند، به لایه‌های بعدی (مثلاً کش دیسک یا کش توزیع‌شده) مراجعه می‌شود.

هر لایه کش با ظرفیت‌های مختلف عمل می‌کند:

کش L1 معمولاً سریعترین است و در حافظه‌های محلی ذخیره می‌شود.

کش L2 سرعت کمتری نسبت به L1 دارد ولی بیشتر می‌تواند داده‌ها را ذخیره کند.

کش L3 معمولاً بزرگتر است و ممکن است بین چندین پردازنده یا سرور به اشتراک گذاشته شود.

کاهش زمان دسترسی به داده‌ها: با استفاده از کش‌های مختلف، داده‌ها به‌طور مؤثرتر ذخیره و بازیابی می‌شوند، که باعث کاهش زمان تأخیر و افزایش کارایی سیستم می‌شود.

مفاهیم

Serial Forwarding Multicast

Serial Forwarding Multicast یک روش انتقال داده در شبکه‌های Multicast است که در آن داده‌ها به ترتیب serial به چندین گیرنده ارسال می‌شوند، به‌جای اینکه به طور هم‌زمان parallel به تمامی گیرندگان ارسال شوند.

ارسال ترتیبی Serial: به‌جای ارسال هم‌زمان داده‌ها به چندین گیرنده، داده‌ها به ترتیب به هر گیرنده ارسال می‌شوند.

Multicast: برخلاف Unicast که در آن داده‌ها به یک گیرنده ارسال می‌شود یا Broadcast که داده‌ها به تمام گیرندگان ارسال می‌شود، در Multicast داده‌ها تنها به یک گروه خاص از گیرندگان ارسال می‌شود.

کاربرد خاص: این روش معمولاً در شرایطی که نیاز به کنترل دقیق یا ترتیب در ارسال داده‌ها وجود دارد استفاده می‌شود.

مفاهیم

Serial Forwarding Multicast

۱- کاهش بار شبکه: در شرایطی که ارسال داده‌ها به طور هم‌زمان برای تمام گیرندگان ممکن است شبکه را بارگذاری کند، Serial Forwarding Multicast می‌تواند به کاهش بار شبکه کمک کند، زیرا داده‌ها به صورت ترتیبی ارسال می‌شوند.

۲- مدیریت دقیق ترتیب ارسال: این روش ممکن است در کاربردهایی که نیاز به ترتیب مشخص و پیش‌بینی‌شده در ارسال داده‌ها دارند، مفید باشد.

۳- کاهش تأخیر در شبکه: ارسال داده‌ها به یک گیرنده در هر بار می‌تواند در مواردی که تأخیر شبکه باید به حداقل برسد، مفید باشد.

مفاهیم پایه
Serverless Distributed Systems

مفاهیم

DistServe

DistServe که ممکن است به صورت Distributed Service یا Distributed Server اختصار داشته باشد، یک مدل معماری است که به **خدمات توزیع‌شده** اشاره دارد.

این مدل در زمینه‌های مختلف فناوری، به ویژه در پردازش ابری، سیستم‌های توزیع‌شده، و مدل‌های مقیاس‌پذیر، برای بهبود عملکرد، مقیاس‌پذیری و مقاومت در برابر خطاها استفاده می‌شود.

DistServe به مفهوم توزیع خدمات در سراسر چندین سرور، نود یا مکان جغرافیایی مختلف اشاره دارد.

این سیستم به‌طور معمول در سیستم‌های توزیع‌شده، ابری یا مقیاس‌پذیر برای بهینه‌سازی فرآیندهای درخواست-پاسخ استفاده می‌شود.

چطور DistServe کار می‌کند؟

در معماری DistServe، درخواست‌ها به‌طور دینامیک به سرورهای مختلف توزیع می‌شوند.

این توزیع ممکن است به‌صورت خودکار با استفاده از الگوریتم‌های زمان‌بندی یا توزیع بار صورت گیرد.

توزیع بار Load Balancing :

بار درخواست‌ها از طریق Load Balancer به سرورهای مختلف توزیع می‌شود. این الگوریتم می‌تواند بر اساس فاکتورهایی مانند ظرفیت سرور، زمان پاسخ‌دهی، یا شلوغی سرور تصمیم‌گیری کند.

محاسبات توزیع‌شده:

محاسبات پردازشی که نیاز به منابع زیادی دارند، به صورت توازی در سرورهای مختلف انجام می‌شوند، که به افزایش کارایی و کاهش زمان پردازش کمک می‌کند.

هماهنگی و همگام‌سازی:

سیستم‌های توزیع‌شده باید هماهنگی و همگام‌سازی دقیقی بین سرورها داشته باشند تا از ایجاد تناقض‌ها یا تداخل داده‌ها جلوگیری شود. این فرآیند معمولاً توسط پروتکل‌های توزیع‌شده انجام می‌شود.

DistServe چه کاربردهایی دارد؟

سیستم‌های ابری:

پلتفرم‌های ابری مانند Amazon Web Services (AWS)، Microsoft Azure و Google Cloud از مدل‌های توزیع‌شده مانند DistServe برای ارائه خدمات مقیاس‌پذیر و مقاوم در برابر خطا استفاده می‌کنند.

مدل‌های یادگیری ماشین:

در سیستم‌های یادگیری ماشین توزیع‌شده، داده‌ها و مدل‌ها بین نودهای مختلف توزیع می‌شوند تا آموزش مدل‌ها و پیش‌بینی‌ها سریع‌تر انجام شود.

سیستم‌های وب مقیاس‌پذیر:

در سیستم‌های وب که نیاز به پردازش تعداد زیادی درخواست دارند، استفاده از DistServe می‌تواند باعث بهبود عملکرد و کاهش زمان تأخیر شود.

مفاهیم

Fine-grained Scaling Abstraction

در Fine-grained scaling abstraction، فرآیند مقیاس‌دهی منابع به‌طور **دقیق و جزئی** انجام می‌شود.

به جای این که کل سیستم یا یک سرویس به‌طور همزمان مقیاس‌دهی شود، می‌توان مقیاس‌دهی را به قسمت‌های کوچکتر یا لایه‌های مختلف تقسیم کرد.

برای مثال:

شما می‌توانید برای هر لایه از سیستم یا بخش خاص (مثل دیتابیس، API، پردازش داده‌ها) به‌طور جداگانه منابع (مانند پردازنده‌ها، حافظه، یا پهنای باند شبکه) تخصیص دهید.

این امکان به شما اجازه می‌دهد که سیستم خود را به شکلی دقیق‌تر و بهینه‌تر مقیاس دهید تا منابع را بر اساس نیاز واقعی هر بخش از سیستم تخصیص دهید.

Coarse-grained Scaling

Coarse-grained scaling معمولاً به مقیاس‌دهی کلی و عمومی سیستم اشاره دارد که در آن منابع به‌طور همزمان و برای کل سیستم یا سرویس تخصیص می‌یابد.

برای مثال، ممکن است منابع برای کل اپلیکیشن یا یک سرویس خاص به‌طور یکسان تخصیص داده شود.

در مقابل، fine-grained scaling به شما این امکان را می‌دهد که دقیقاً برای هر بخش، سرویس یا لایه منابع را تخصیص دهید.

مفاهیم

Latency-aware scheduling

Latency-aware scheduling یک رویکرد در زمان‌بندی scheduling است که در آن زمان تأخیر latency یکی از فاکتورهای اصلی در تصمیم‌گیری برای تخصیص منابع یا برنامه‌ریزی اجرای وظایف است.

این نوع زمان‌بندی معمولاً در سیستم‌های توزیع‌شده، شبکه‌های پردازشی و محاسبات ابری استفاده می‌شود، جایی که تاخیر در پردازش داده‌ها یا ارسال درخواست‌ها می‌تواند بر عملکرد و تجربه کاربری تأثیر قابل توجهی بگذارد.

تصمیمات مربوط به زمانبندی تخصیص منابع یا اجرای وظایف بر اساس زمان تأخیر اتخاذ می‌شود.

Latency-aware scheduling چگونه کار میکند؟

۱- شناسایی وظایف و منابع:

ابتدا سیستم باید وظایف مختلف و منابع موجود (سرورها، کانتینرها، شبکه‌ها) را شناسایی کند.

۲- اندازه‌گیری تأخیر:

سپس سیستم باید تأخیر مربوط به اجرای هر وظیفه در منابع مختلف را اندازه‌گیری کرده و اطلاعات مربوط به زمان تأخیر را برای هر مسیر پردازشی یا ارتباطی ذخیره کند.

۳- تخصیص بهینه وظایف:

پس از تجزیه و تحلیل، سیستم تصمیم می‌گیرد که کدام وظیفه باید روی کدام منابع اجرا شود تا تأخیر به حداقل برسد. این تصمیمات ممکن است شامل انتخاب بهترین سرور، پهنای باند شبکه یا نودهای پردازشی باشند.

۴- مدیریت زمان پاسخ:

در نهایت، سیستم وظایف را به گونه‌ای زمان‌بندی می‌کند که زمان پاسخ‌دهی به درخواست‌ها یا پردازش‌ها به حداقل برسد.

مفاهیم پایه
Hardware Networking

مفاهیم

RDMA Remote Direct Memory Access

Remote Direct Memory Access (RDMA) به پروتکلی گفته می‌شود که امکان انتقال داده‌ها را بین دستگاه‌ها (مثل سرورها یا کارت‌های شبکه) به‌طور مستقیم در حافظه فراهم می‌آورد.

در این فناوری، داده‌ها از حافظه یک دستگاه به حافظه دستگاه دیگر منتقل می‌شوند بدون اینکه نیاز به پردازش‌های اضافی از سمت پردازنده CPU یا سیستم‌عامل باشد.

این امر باعث می‌شود که انتقال داده‌ها بسیار سریع‌تر و با تأخیر کمتر از روش‌های سنتی مانند TCP/IP باشد.

مفاهیم

RDMA Remote Direct Memory Access

در RDMA، ارتباط میان دو دستگاه (مثلاً دو سرور) از طریق شبکه‌های سریع (مانند InfiniBand یا RDMA over Converged Ethernet (RoCE) برقرار می‌شود.

این فرآیند به این صورت است:

۱- **درخواست RDMA :** یکی از دستگاه‌ها درخواست می‌کند که داده‌ها به‌طور مستقیم از حافظه دستگاه دیگر خوانده یا نوشته شوند.

۲- **انتقال داده‌ها**: پس از تایید درخواست، داده‌ها بدون نیاز به پردازش از طریق شبکه‌های خاص InfiniBand یا RoCE از حافظه یک دستگاه به حافظه دستگاه دیگر منتقل می‌شود.

۳- **بازخورد به پردازنده**: پس از اتمام انتقال، هیچ پردازش اضافی از طرف CPUیا سیستم‌عامل انجام نمی‌شود و پردازنده‌ها تنها مسئولیت مدیریت کنترل‌های دسترسی و هماهنگی‌ها را بر عهده دارند.

مزایای RDMA

کاهش تأخیر Low Latency : انتقال داده‌ها بدون نیاز به پردازش توسط سیستم‌عامل یا CPU، باعث کاهش تأخیر می‌شود. این ویژگی برای سیستم‌هایی که نیاز به پاسخ سریع دارند، مانند پردازش‌های بلادرنگ و مقیاس‌پذیری بالا بسیار مهم است.

افزایش کارایی: RDMA به کاهش بار CPU کمک می‌کند و این امکان را فراهم می‌آورد که پردازنده‌ها بیشتر روی وظایف محاسباتی تمرکز کنند تا انتقال داده‌ها.

مقیاس‌پذیری بالا: RDMA برای سیستم‌های توزیع‌شده و مقیاس‌پذیر به‌ویژه در دیتا سنترها و برنامه‌های ابری مفید است. این تکنولوژی می‌تواند در مقیاس‌های بزرگ داده‌ها را با سرعت بالا و تأخیر کم منتقل کند.

کارایی بالا برای بارهای سنگین داده: RDMA به‌ویژه برای انتقال داده‌های حجیم و بارهای سنگین مانند پردازش داده‌های عظیم و ذخیره‌سازی داده‌های توزیع‌شده مؤثر است.

مفاهیم

NVLink

NVLink یک فناوری ارتباطی با باند پهنای بالا است که توسط NVIDIA توسعه یافته و برای ارتباط بین پردازنده‌های گرافیکی GPU طراحی شده است.

این فناوری به سیستم‌های پردازش موازی کمک می‌کند تا عملکرد بالا، مقیاس‌پذیری بهتر و کاهش تأخیر را در سیستم‌های محاسباتی فراهم کنند.

با استفاده از NVLink، می‌توان مدل‌های یادگیری عمیق، شبیه‌سازی‌های علمی و رندرینگ‌های پیچیده را به‌طور مؤثرتر و سریع‌تر انجام داد.

مزایای NVLink

۱- عملکرد بیشتر در پردازش‌های موازی: NVLink این امکان را می‌دهد که چندین GPU در یک سیستم با یکدیگر همکاری کنند، که باعث افزایش سرعت پردازش داده‌ها می‌شود.

این امر در یادگیری ماشین و شبیه‌سازی‌های علمی که به قدرت پردازشی زیادی نیاز دارند، ضروری است.

۲- مقیاس‌پذیری بالا برای سیستم‌های پردازشی بزرگ: استفاده از NVLink برای اتصال چندین GPU در یک سیستم به راحتی امکان‌پذیر است و این امکان را می‌دهد که سیستم‌های پردازشی مقیاس‌پذیر با عملکرد بالا ایجاد شوند.

۳- سریع‌تر از PCIe : NVLink به‌طور قابل توجهی از PCIe سریع‌تر است. در حالی که PCIe برای اکثر سیستم‌ها مناسب است، NVLink به دلیل پهنای باند بالاتر، گزینه‌ای عالی برای کارهایی است که نیاز به انتقال سریع و موازی داده‌ها دارند.

مفاهیم

TPU Tensor Processing Unit

مدل‌های یادگیری عمیق Deep Learning مثل شبکه‌های عصبی، حجم زیادی از محاسبات ماتریسی و عددی دارند.

CPU ها و حتی GPU ها که برای محاسبات عمومی یا گرافیکی طراحی شده‌اند، برای این کار بهینه نیستند.

بنابراین گوگل تصمیم گرفت یک پردازنده خاص بسازد که:

۱- فقط روی عملیات‌های مربوط به Tensor مثل ضرب ماتریس‌ها تمرکز کند

۲- سرعت و بهره‌وری بالاتری نسبت به CPU و GPU برای مدل‌های یادگیری ماشین داشته باشد

۳- مصرف برق کمتری داشته باشد

Tensor Processing Unit (TPU)

- Proposal: Design a custom ASIC for the inference phase of NN (training still happens using GPUs)
- Principles:
  - improve cost-performance by 10X compared to GPUs
  - simple design for response time guarantees (single-thread, no prefetching, no OOO etc)
- Characteristics:
  - More like a co-processor to reduce time-to-market delays
    - Host sends instructions to TPU
  - connected through PCIe I/O bus

Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

Compute Fabric

Compute Fabric به طور کلی به یک شبکه یا لایه ارتباطی گفته می‌شود که منابع محاسباتی مختلف را به یکدیگر متصل می‌کند و این امکان را فراهم می‌آورد که این منابع به‌طور یکپارچه و به‌طور مؤثر به اشتراک گذاشته شوند.

این معماری به خصوص در سیستم‌های مقیاس‌پذیر و محاسبات توزیع‌شده استفاده می‌شود، جایی که نیاز به مدیریت هماهنگ منابع پردازشی و ذخیره‌سازی وجود دارد.

Compute Fabric چگونه کار میکند؟

۱- اتصال منابع پردازشی به شبکه:

منابع مختلف محاسباتی (مانند CPU، GPU و دستگاه‌های ذخیره‌سازی) به یک شبکه بزرگتر متصل می‌شوند که امکان دسترسی سریع به داده‌ها و منابع بین دستگاه‌ها را فراهم می‌آورد.

۲- مدیریت منابع و بارهای پردازشی:

سیستم Compute Fabric به‌طور پویا منابع را تخصیص می‌دهد و آن‌ها را بر اساس نیازهای خاص بار پردازشی تخصیص می‌دهد.

این شامل استفاده از میکروسرویس‌ها، پردازش موازی و مدیریت مقیاس‌پذیر است.

Compute Fabric چگونه کار میکند؟

۳- پردازش‌های موازی و توزیع‌شده:

پردازش‌های بزرگ به‌طور موازی در نودهای مختلف انجام می‌شوند و داده‌ها بین نودها به‌طور مؤثر توزیع می‌شوند تا بار کاری کاهش یابد و سرعت پردازش افزایش یابد.

۴- اتصال سریع داده‌ها:

از آنجا که داده‌ها باید در زمان کم انتقال یابند، این معماری معمولاً از شبکه‌های سریع (مانند RDMA یا InfiniBand) برای انتقال داده‌ها و به حداقل رساندن تأخیر استفاده می‌کند.

مفاهیم

Micro Service

در معماری میکروسرویس‌ها، یک برنامه بزرگ به مجموعه‌ای از سرویس‌های کوچک و مستقل تقسیم می‌شود.

هر میکروسرویس معمولاً مسئول انجام یک عملکرد خاص است و می‌تواند به‌طور مستقل توسعه داده شده، آزمایش شود، به‌روزرسانی شود و مقیاس یابد.

این سرویس‌ها از طریق رابط‌های برنامه‌نویسی اپلیکیشن API یا پیام‌ها با یکدیگر ارتباط برقرار می‌کنند.

مفاهیم

Incast Traffic

به وضعیتی در شبکه اشاره دارد که در آن تعداد زیادی سرور یا نودها (در یک شبکه توزیع‌شده یا دیتاسنتر) همزمان داده‌هایی را به یک سرور مقصد واحد ارسال می‌کنند، که باعث ایجاد تراکم congestion و کاهش پهنای باند شبکه می شود.

در یک سیستم پردازش توزیع‌شده که در آن چندین سرور اطلاعات خود را به یک سرور ذخیره‌سازی مرکزی ارسال می‌کنند، این سیستم ممکن است با Incast Traffic روبرو شود، جایی که همه سرورها به‌طور همزمان داده‌ها را به یک سرور مقصد ارسال می‌کنند و در نتیجه ترافیک شبکه افزایش می‌یابد و باعث کندی در سیستم می‌شود.

مفاهیم

Outcast Traffic:

ترافیکی است که در آن یک سرور یا نود داده‌ای را به چندین گیرنده ارسال می‌کند.

این وضعیت معمولاً کمتر از Incast باعث تراکم می‌شود اما می‌تواند در سیستم‌هایی که نیاز به توزیع داده‌ها به تعداد زیادی گیرنده دارند مشکلاتی ایجاد کند.

در یک شبکه پخش ویدئو که داده‌های ویدئویی از یک سرور به چندین گیرنده ارسال می‌شود، می‌تواند به Outcast Traffic منجر شود.

در اینجا، ممکن است سرور نتواند داده‌ها را به همه گیرندگان به‌طور مؤثر توزیع کند، که باعث تأخیر یا افت کیفیت پخش شود.

مفاهیم

تفاوت بین Outcast Traffic و Incast

ویژگی	Incast Traffic	Outcast Traffic
نوع ترافیک	چندین سرور به یک سرور ارسال می‌کنند	یک سرور به چندین سرور یا گیرنده ارسال می‌کند
مشکل اصلی	**تراکم در گیرنده مقصد** و **اختناق شبکه**	**تراکم در ارسال کننده** و **ازدحام در مسیرهای ارسال**
مثال‌های کاربردی	سیستم‌های ذخیره‌سازی توزیع‌شده، پردازش‌های موازی	پخش ویدئو، توزیع بار، سیستم‌های سرور مرکزی
تأثیر بر عملکرد	**کاهش عملکرد شبکه** به دلیل تراکم داده‌ها	**کاهش عملکرد در ارسال‌کننده** به دلیل فشار زیاد

مفاهیم

Chain Scheduling

در Chain Scheduling، مجموعه‌ای از وظایف به صورت زنجیره‌ای زمان‌بندی و اجرا می‌شوند.

این وظایف به گونه‌ای تنظیم می‌شوند که پردازش هر کدام به پردازش بعدی وابسته باشد و معمولاً به صورت سری یا پیوسته اجرا می‌شوند.

در این مدل زمان‌بندی، هر وظیفه زمانی که انجام شد، داده‌ها یا نتایج خود را به وظیفه بعدی منتقل می‌کند به‌طوری که خروجی یک وظیفه به ورودی وظیفه بعدی تبدیل می‌شود.

این روش بیشتر در سیستم‌های توزیع‌شده و پردازش‌های موازی استفاده می‌شود، به ویژه زمانی که نیاز است که پردازش‌ها یا کارها به‌طور منطقی و به ترتیب انجام شوند.

بررسی مقاله دوم

Fast and Live Model Auto Scaling with O(1) Host Caching

مقیاس‌پذیری سریع و زنده مدل با کش‌سازی O(1) (بلادرنگ) در میزبان

چکیده

مقیاس‌پذیری خودکار مدل، یکی از ارکان اصلی در ارائه سرویس مدل‌محور به‌صورت بدون سرور Serverless است. با این حال، این فرآیند با چالش مهمی مواجه است: باید بین سرعت مقیاس‌پذیری و میزان استفاده از حافظه/فضای ذخیره‌سازی برای کش کردن پارامترهای مدل، تعادل برقرار کرد. این تعادل باعث می‌شود که سیستم نتواند به‌خوبی پاسخگوی نیاز به مقیاس‌پذیری سریع و مکرر در میان چندین میزبان باشد. مشکل اصلی، عملکرد کند سطح داده Data Plane است؛ چراکه نمونه‌های مقیاس‌یافته تا زمان بارگذاری کامل پارامترها غیرفعال می‌مانند.

در این مقاله نشان داده شده است که می‌توان بدون یا با کمترین نیاز به کش کردن در حد $O(1)$، داده‌ها را از طریق شبکه‌ی محاسباتی بین GPUها بارگذاری کرد. دلیل این امر، دو نکته است: (۱) سرعت شبکه‌ی بین GPUها تقریباً با کش میزبان برابر است و معمولاً به‌طور کامل استفاده نمی‌شود؛ و (۲) بارگذاری همزمان چند نمونه از مدل‌ها می‌تواند با استفاده از تکنیک چندپخشی Multicast در شبکه، بدون نیاز به کش‌های سنگین انجام شود.

همچنین، برای زنده‌کردن فرآیند مقیاس‌پذیری (یعنی قابل استفاده بودن نمونه‌ها حتی قبل از پایان بارگذاری)، یک روش جدید معرفی شده که در آن به جای مقیاس‌پذیری در سطح کل نمونه، از مقیاس‌پذیری لایه‌به‌لایه استفاده می‌شود. در این روش، بخش‌هایی از محاسبات مدل از نمونه‌های پرکار به نمونه‌های جدید منتقل می‌شود و در نتیجه حتی اگر سرعت شبکه کافی نباشد، باز هم امکان خدمت‌رسانی حفظ می‌شود.

سامانه پیشنهادی ما، به نام BLITZSCALE، تا ۸۶٪ زمان تأخیر سرویس‌دهی را کاهش می‌دهد و عملکردی برابر یا حتی بهتر از حالتی دارد که تمام پارامترهای مدل از پیش کش شده‌اند.

مقاله

اهداف

افزایش سرعت مقیاس‌پذیری خودکار مدل‌ها

کاهش وابستگی به کش میزبان Host Cache

پیاده‌سازی مقیاس‌پذیری زنده Live Autoscaling

طراحی و پیاده‌سازی سیستم BLITZSCALE

کاهش تأخیر در پاسخ‌دهی مدل‌ها TTFT و TBT

بهینه‌سازی مصرف منابع سخت‌افزاری (GPU، حافظه، شبکه)

مقاله

سؤال اصلی مقاله:

چگونه می‌توان مقیاس‌پذیری خودکار مدل‌های یادگیری عمیق را به‌صورت سریع و زنده Live، بدون وابستگی سنگین به کش میزبان، در زیرساخت‌های مدل‌محور مانند MAAS پیاده‌سازی کرد؟

چالش اصلی در سیستم‌های مدل به‌عنوان‌سرویس MAAS:

ترکیب سرعت بالا در مقیاس‌پذیری با استفاده بهینه از منابع و بدون نیاز به ذخیره‌سازی سنگین پارامترها در حافظه یا SSD .

راه‌حل پیشنهادی مقاله BLITZSCALE

۱- بارگذاری پارامترهای مدل از طریق شبکه محاسباتی بین GPUها

استفاده از شبکه‌های پرسرعت مانند RDMA یا NVLink برای انتقال مستقیم پارامترها، به‌جای بارگذاری از SSD یا DRAM میزبان.

راه‌حل پیشنهادی مقاله BLITZSCALE

۲- کاهش نیاز به کش از طریق O(1) Host Caching

تنها یک نسخه از مدل روی یک میزبان نگهداری می‌شود و از آن برای بارگذاری به سایر نمونه‌ها استفاده می‌گردد کش در مقیاس O(1) .

راه‌حل پیشنهادی مقاله BLITZSCALE

۳- استفاده از تکنیک چندپخشی Multicast در شبکه برای انتقال سریع پارامترها

ارسال همزمان پارامترها از یک منبع به چندین مقصد (نه همه) بدون نیاز به ذخیره‌سازی محلی در همه میزبان‌ها.

راه‌حل پیشنهادی مقاله BLITZSCALE

۴- مقیاس‌پذیری زنده Live Autoscaling از طریق اجرای لایه‌ای مدل‌ها

مدل‌ها به صورت لایه‌به‌لایه اجرا می‌شوند؛ به طوری که نمونه‌های جدید می‌توانند بخشی از محاسبات را انجام دهند حتی قبل از بارگذاری کامل مدل.

راه‌حل پیشنهادی مقاله BLITZSCALE

۵- زمان‌بندی پیشرفته درخواست‌ها با الگوریتم Zigzag Scheduling

استفاده از زمان‌بندی تطبیقی برای هماهنگی بین نمونه‌های پرکار و نمونه‌های در حال بارگذاری، با هدف کاهش تأخیر پاسخ‌دهی TTFT و TBT.

راه‌حل پیشنهادی مقاله BLITZSCALE

۶- برنامه‌ریزی هوشمند برای جلوگیری از تداخل ترافیک شبکه با درخواست‌های سرویس‌دهی

انتخاب مسیرهای انتقال پارامتر به گونه‌ای که باعث اختلال در جریان داده‌های جاری سرویس‌دهی نشوند.

مقاله

طراحی یک برنامه‌ریز مقیاس‌پذیری آگاه از مدل model-aware scale planner که:

۱- الگوی جریان داده در مدل را می‌شناسد

مثلاً می‌داند در پیش‌پردازش prefill داده به چه سمتی می‌رود، و در decode چه مسیرهایی درگیر هستند.

۲- مسیریابی انتقال پارامترها را طوری انجام می‌دهد که با مسیرهای درگیر در سرویس‌دهی هم‌پوشانی نداشته باشند

یعنی از منابعی برای ارسال استفاده می‌کند که در حال حاضر خلوت یا آزاد هستند.

طراحی یک برنامه‌ریز مقیاس‌پذیری آگاه از مدل model-aware scale planner که:

۳- برنامه‌ریزی سریالی Serial Forwarding Chain به جای ارسال موازی

با استفاده از زنجیره‌های چندپخشی سریالی، پارامترها مرحله به مرحله منتقل می‌شوند بدون اینکه کل شبکه را اشباع کنند.

استفاده هوشمندانه از شبکه دوطرفه Bi-directional Network

شبکه دوطرفه چیست؟

شبکه‌های مدرن دیتاسنتر (مانند RDMA بین GPUها) معمولاً دوطرفه full-duplex هستند؛ یعنی می‌توان هم‌زمان در دو جهت مخالف داده ارسال کرد:

از GPU A به GPU B

و به‌طور هم‌زمان از GPU B به GPU A

بدون اینکه این دو جریان مزاحم یکدیگر شوند (تا حدی مشخص و در صورت مدیریت صحیح).

استفاده هوشمندانه از شبکه دوطرفه Bi-directional Network

شبکه دوطرفه چیست؟

مثال ساده:

فرض کن در سرویس‌دهی معمول، داده‌های میانی از Prefill به Decode ارسال می‌شوند جهت $A \rightarrow B$.

حال اگر در همان مسیر $A \rightarrow B$ بخواهیم پارامترهای مدل را هم در مقیاس‌پذیری منتقل کنیم، این دو ترافیک با هم تداخل می‌کنند.

اما اگر پارامترها را از $B \rightarrow A$ منتقل کنیم، از جهت آزاد شبکه استفاده شده و تداخل رخ نمی‌دهد.

مقاله

در سیستم‌های قبلی مثل ServerlessLLM، (مقاله اول) برای اینکه مقیاس‌پذیری سریع باشد، باید پارامترهای مدل از قبل در حافظه میزبان Host Cache همه نودها کش شده باشند.

اما این کار:

حافظه زیادی می‌خواهد (غیرمقیاس‌پذیر)

کش‌های زیادی نیاز دارد (غیرعملی برای مدل‌های بزرگ)

باز هم در صورت miss شدن کش، کند عمل می‌کند

مقاله

اما این مقاله می‌گوید:

نیازی نیست پارامترهای مدل روی همه نودها موجود باشد. فقط کافی‌ست در حد O(1) یعنی یک یا چند کپی محدود از مدل در خوشه cluster وجود داشته باشد.

و از آن استفاده می‌شود برای:

Multicast چندپخشی به نودهای دیگر

یا Serial Forwarding Chain برای انتشار پارامترها از یک نود به سایر نودها از طریق GPUها

اما این مقاله می‌گوید:

به‌جای تلاش برای کش کردن مدل در همه نودها (که عملاً ممکن نیست)، فقط یک نسخه از مدل در خوشه نگه‌داری می‌شود (مثلاً در حافظه CPUیک ماشین).

این نسخه همیشه در دسترس است، بنابراین حتی در شرایط cold startکامل، مدل می‌تواند سریع‌تر از حافظه CPUیا GPUدیگر پخش شود و نیازی به خواندن از SSD که کندتر است نیست.

اگر حتی یک نسخه از مدل در سیستم موجود باشد:

با استفاده از شبکه‌های پرسرعت (مثل RDMA 100–200 Gbpsیا NVLinkتا ۱.۶ Tbps) می‌توان پارامترها را از آن نسخه به سایر نودها منتقل کرد.

این کار با استفاده از Serial Forwarding Chainیا Multicast Plan به‌صورت زنجیره‌ای انجام می‌شود، که نسبت به روش سنتی بارگزاری از دیسک یا SSD بسیار سریع‌تر است.

نقاط ضعف:

۱- فرض وجود حداقل یک نسخه از مدل O(1) caching

مقاله فرض می‌کند همیشه حداقل یک نسخه از مدل (مثلاً در حافظه CPUیک نود) در خوشه وجود دارد.

در عمل، اگر تمام نسخه‌ها پاک شوند مثلاً به‌دلیل scale-down یا crash، cold start واقعی رخ می‌دهد و باید از SSD بارگذاری شود که بسیار کند است.

مقاله هیچ راه‌حلی برای تضمین در دسترس بودن مداوم آن نسخه ارائه نمی‌دهد جز یک سیاست ساده initial load .

پیشنهاد بهبود: استفاده از replication هوشمند برای تضمین در دسترس بودن نسخه‌ها.

نقاط ضعف:

۲- پیچیدگی اجرای شبکه‌ای multicast planning

الگوریتم‌های multicast مثل chain-based forwarding در مقاله، اگرچه سریع‌اند، ولی:

در شبکه‌های ناهمگن و پویا ممکن است ناکارآمد باشند.

برنامه‌ریزی زنده online برای انتخاب مسیر بهینه NP-hard است، و مقاله فقط از یک روش greedy استفاده کرده که همیشه بهینه نیست.

در شرایطی که ترافیک شبکه بالاست، interferenceبا workload servingممکن است مشکل‌زا شود.

پیشنهاد بهبود: ادغام با SDN (Software Defined Networking) برای کنترل بهتر مسیرهای شبکه‌ای.

با تشکر از همراهی شما

محمد سعید صفایی صادق