تخصیص منابع مبتنی بر مکانیسم *Gossip* برای محاسبات سبز در ابرهای بزرگ

چکیده

ما به مسئله تخصیص منابع در محیط ابر در مقیاس بزرگ می‌پردازیم، که بهینه‌سازی پیکربندی ابر به‌صورت پویا برای اهداف محاسبات سبز تحت محدودیت‌های پردازنده و حافظه را رسمی می‌کنیم. ما پروتکل عمومی *gossip* را برای تخصیص منابع پیشنهاد می‌کنیم، که می‌تواند برای اهداف خاص معرفی شود. ما نمونه‌ای از این پروتکل عمومی را باهدف به حداقل رساندن مصرف انرژی از طریق تحکیم[1] سرور توسعه می‌دهیم، درحالی‌که رضایت‌مندی تغییر الگوی بارگیری برآورده شود. این پروتکل، *GRMP-Q* نامیده شد، که یک راه‌حل ابتکاری[2] کارآمد فراهم می‌کند که در اغلب موارد به خوبی عمل می‌کند— در موارد خاص، بهینه است. تحت سربار[3]، پروتکل یک تخصیص عادلانه از منابع پردازنده به خدمات گیرنده[4] را می‌دهد. نتایج شبیه‌سازی نشان می‌دهد که معیارهای کلیدی عملکرد، با افزایش اندازه سیستم تغییر نمی‌کند، فرآیند تخصیص منابع مقیاس‌پذیر برای بالاتر از 100000 سرور. به‌طورکلی، اثربخشی پروتکل در دستیابی به اهداف خود، با افزایش ظرفیت حافظه در سرورها، افزایش می‌یابد.

کلمات کلیدی: محاسبات ابر، محاسبات سبز، مدیریت توزیع، مدیریت انرژی، تخصیص منابع، پروتکل *gossip*، تحکیم سرور

[1] consolidation
[2] Heuristic
[3] Overload
[4] Client

1. مقدمه

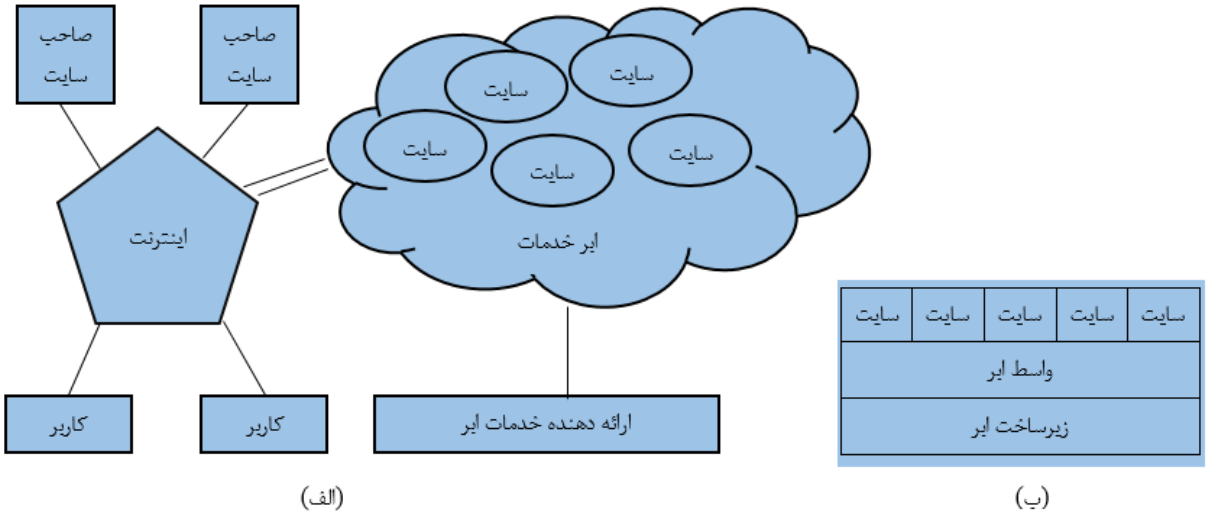
مصرف انرژی در مراکز داده[5] قابل توجه است و به سرعت در سال‌های اخیر رشد کرده است و این رشد انتظار می‌رود ادامه‌دار باشد. چند مطالعه آن را نشان می‌دهد [1]-[3]. یک روش مؤثر برای کاهش مصرف انرژی مراکز داده، تحکیم سرور [4]، [5] است که باهدف تمرکز حجم کار[6] بر روی حداقل تعداد سرورها است. این کار مؤثر است، زیرا امروزه سطح بهره‌برداری در مراکز داده اغلب پایین است، حدود 15٪ [6]. برای یک اجرا، سرور بیش از 60٪ از حداکثر مصرف انرژی خود را مصرف می‌کند، حتی اگر آن بار حمل نکند ([4])، تعویض سرورها (به‌طور موقت) برای یک حالت ضروری نباشد که حداقل یا هیچ انرژی نیاز دارد که می‌تواند به‌طور قابل‌توجهی مصرف انرژی را کاهش دهد.

امروزه همه تکنولوژی‌های کلیدی موردنیاز برای تحکیم سرور، در دسترس هستند. تکنولوژی‌های مجازی سازی[7] و مهاجرت زنده[8]، استحکام پویا حجم کار را تحت درخواست پشتیبانی می‌کند. داشتن سطوح مختلف از حالت آماده‌به‌کار[9] (مشخص شده با سطوح مختلف مصرف انرژی و زمان بیدار شدن[10]) که پیشنهادهای تجهیزات مدرن، برای انطباق منابع مرکز داده برای نیازهای در حال تغییر را اجازه می‌دهد.

در این کار، ما به مسئله مدیریت منابع برای محیط ابر در مقیاس بزرگ (محدوده بالاتر از 100000 سرور) باهدف خدمت یک حجم کار پویا با حداقل مصرف انرژی می‌پردازیم. درحالی‌که سهم ما در زمینه کلی‌تری است، ما بحث را از منظر مفهوم $PaaS$ [11]، با مورد استفاده[12] خاص از یک ارائه‌دهنده خدمات ابر[13] که سایت‌های میزبان‌ها[14] در محیط ابر است، انجام می‌دهیم. ذینفعان[15] در این مورد استفاده در شکل 1(الف) نشان داده می‌شود. ارائه‌دهنده

[5] Datacenters
[6] Workload
[7] Virtualization
[8] live migration
[9] standby
[10] Wakeup
[11] Platform-as-a-Service
[12] Use case
[13] Cloud service provider
[14] Host
[15] Stakeholder

خدمات ابر، زیر ساخت‌های فیزیکی، که در آن خدمات ابر ارائه شده است را اداره می‌کند. این، سرویس‌های میزبانی[16] به صاحبان سایت‌ها از طریق یک واسط[17] که در زیرساخت‌هایش اجرا می‌شود را ارائه می‌دهد (شکل 2(ب) را ببینید). صاحبان سایت‌ها خدمات را به کاربران مربوطه خود از طریق سایت‌هایی که توسط ارائه‌دهنده خدمات ابر میزبان می‌شوند ارائه می‌دهند.



شکل 1. (الف) سناریو استقرار ذینفعان محیط ابر در نظر گرفته‌شده در این کار. (ب) معماری کلی محیط ابر. این کار بر روی مدیریت منابع انجام شده توسط لایه واسط تمرکز دارد [7].

نتایج حاصل از این کار به مهندسی یک لایه واسط که تخصیص منابع در یک محیط ابر را انجام می‌دهد، با اهداف طراحی زیر کمک می‌کند.

1) هدف عملکرد

a. وقتی سیستم در زیر بار است، هدف به حداقل¬رسانی مصرف انرژی است از طریق تثبیت سرور، درحالی‌که رضایت¬مندی تقاضا از سایت¬های میزبان باشد.

b. وقتی سیستم در سربار است، هدف تخصیص منابع موجود به‌طور عادلانه در تمامی سایت¬های میزبان است.

2) سازگاری: فرآیند تخصیص منابع باید به‌صورت پویا و کارآمد برای تغییرات در تقاضا سازگار باشد.

[16] Hosting
[17] middleware

3) مقیاس¬پذیری: فرآیند تخصیص منابع باید در تعداد سرورهای موجود در ابر و تعداد سایت¬های میزبان ابر مقیاس-پذیر باشد. به‌طور خاص، منابع مصرف شده در هر سرور به‌منظور دستیابی به یک هدف عملکرد معین، باید با تعداد سرورها و تعداد سایت¬ها افزایش یابد.

رویکرد ما در اطراف یک طراحی غیرمتمرکز مرکزیت می‌یابد که در آن اجزای لایه واسط در هر سرور از محیط ابر اجرا می‌شود. (ما در ادامه این مقاله به یک سرور از ابر، به‌عنوان یک دستگاه رجوع می‌کنیم). برای رسیدن به یک مقیاس‌پذیری، ما تصور می‌کنیم که تمام وظایف کلیدی لایه واسط، شامل برآورد حالات عمومی[18]، قرار دادن ماژول-های سایت و محاسبه سیاست‌ها برای درخواست ارسال، براساس الگوریتم‌های توزیع‌شده هستند. بر خلاف نرم افزار مدیریت موجود برای ابرهای خصوصی، مانند *OpenNebula* [8]، *OpenStack* [9]، *AppScale* [10] و *Cloud Foundry* [11]، راه‌حل پیشنهادی ما، در یک شکل ترکیب‌شده و یکپارچه فراهم می‌شود. (الف) انطباق پویا از تخصیص منابع موجود در پاسخ به یک تغییر، (ب) مقیاس‌گذاری پویا منابع برای یک کاربرد آنسوی یک دستگاه فیزیکی تک و (ج) مقیاس‌پذیری آنسوی برخی از 100000 سرور.

این مقاله بر اساس کار قبلی ما در مورد مدیریت منابع مقیاس‌پذیر برای محیط‌های ابر است [7]. آن از معماری واسط از آن کار را استفاده می‌کند، رسمی‌سازی مسئله تخصیص منابع را سازگار می‌کند و دوباره از مفهوم محاسبات سیاست‌های تخصیص منابع از طریق پروتکل *gossip* استفاده می‌کند. سهم کلیدی این مقاله به شرح زیر است. ابتدا ما پروتکل عمومی *gossip* را برای مدیریت منابع در محیط ابر معرفی می‌کنیم که می‌تواند برای اهداف خاص معرفی شود. سپس، ما مسئله حداقل سازی مصرف انرژی از طریق تثبیت سرور را رسمی می‌کنیم و یک راه‌حل ابتکاری در قالب یک نمونه از پروتکل عمومی ارائه می‌دهیم. درنهایت ما از طریق شبیه‌سازی اثربخشی پروتکل در مقایسه با یک سیستم ایده آل نشان می‌دهیم، و ما نشان می‌دهیم که پروتکل برای ابر بسیار بزرگ به خوبی مقیاس می‌کند.

ساختار مقاله به‌صورت زیر است:

[18] Global

بخش 2 معماری یک لایه واسط که مدیریت منابع را برای یک محیط ابر در مقیاس بزرگ انجام می‌دهد را شرح می‌دهد. بخش 3 مدل ما برای مدیریت منابع در محیط ابر و راه‌حل عمومی برای مسئله مدیریت منابع را ارائه می‌دهد. بخش 4 مسئله خاص موردمطالعه در این مقاله و راه‌حل ما را ارائه می‌دهد. راه‌حل از طریق شبیه‌سازی در بخش 5 ارزیابی می‌شود. بخش 6 کارهای مشابه را مرور می‌کند و بخش 7 نتیجه‌گیری از این مقاله و شرح کارهای آینده را شامل می‌شود.

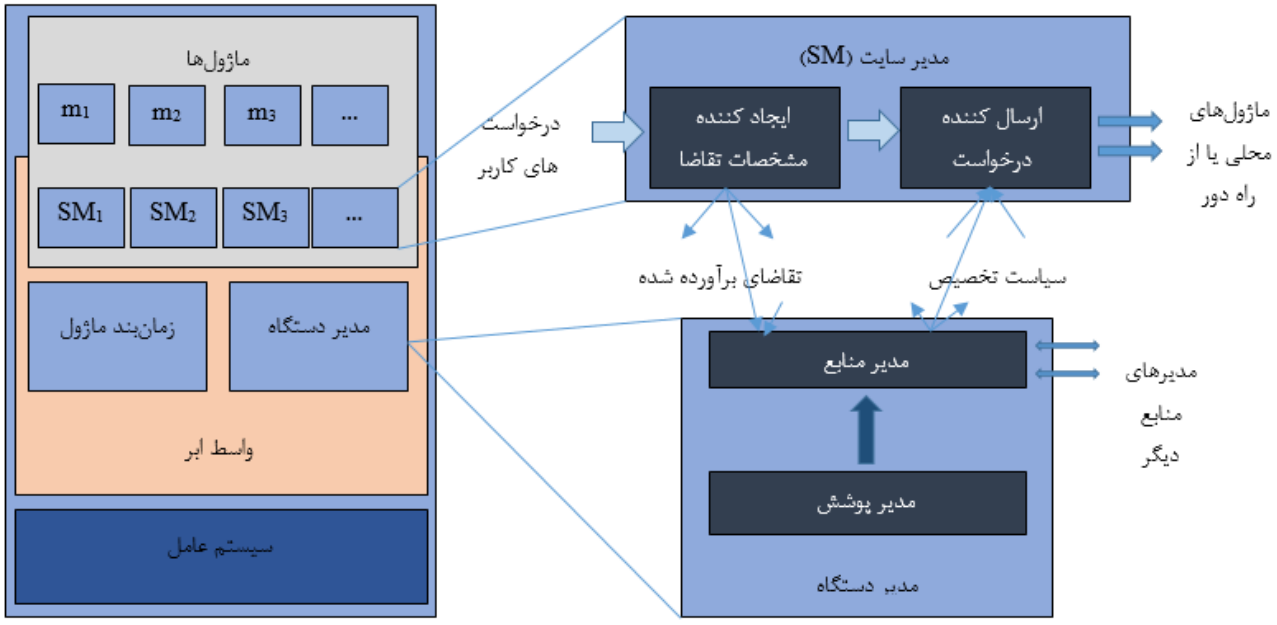
2. معماری سیستم

شکل 2 (سمت چپ) معماری واسط ابر را نشان می‌دهد. مؤلفه‌های لایه واسط روی تمام دستگاه‌ها اجرا می‌شود. منابع ابر در درجه اول توسط نمونه‌های ماژول مصرف می‌شوند که به موجب آن قابلیت‌های یک سایت از یک یا چند ماژول ساخته شده است. واسط، شامل بخشی از منطق خدمات یک سایت (نشان داده‌شده با m_i در شکل 2) یا مدیر سایت (نشان داده‌شده با SM_i) است.

هر دستگاه یک مؤلفه مدیر دستگاه که سیاست تخصیص منابع را محاسبه می‌کند را اجرا می‌کند، که شامل تصمیم‌گیری نمونه‌های ماژول برای اجرا می‌شود. سیاست تخصیص منابع توسط یک پروتکل (در این مقاله $GRMP$ نامیده می‌شود) محاسبه می‌شود که در مؤلفه مدیر منابع اجرا می‌شود. این مؤلفه تقاضای طرح‌ریزی شده را برای هر ماژول به‌عنوان ورودی می‌گیرد که دستگاه اجرا می‌کند. سیاست تخصیص محاسبه‌شده، به زمان‌بند ماژول برای اجرا و همچنین مدیر سایت برای تصمیم‌گیری درباره درخواست ارسال، ارسال می‌شود. مدیر پوشش[19]، یک الگوریتم توزیع‌شده را اجرا می‌کند که یک نمودار پوشش[20] از دستگاه در ابر را حفظ می‌کند و هر مدیر منبع یک لیست دستگاه‌ها برای تعامل فراهم می‌کند.

[19] Overlay manager
[20] Overlay graph

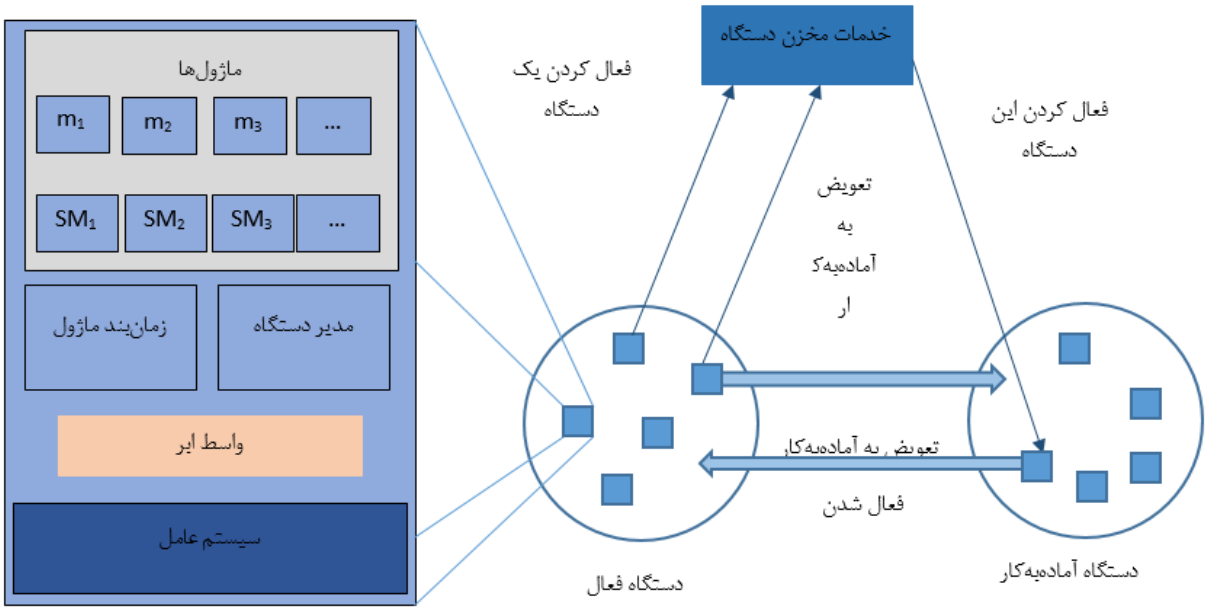
معماری ما یک مدیر سایت را با هر سایت مرتبط می‌سازد. هر مدیر سایت، درخواست کاربران به یک سایت خاص را کنترل می‌کند. این دو مؤلفه مهم دارد: ایجادکننده مشخصات[21] تقاضا و ارسال‌کننده درخواست. ایجادکننده مشخصات تقاضا، تقاضای منابع از هر ماژول سایت بر اساس آمار درخواست، اهداف کیفیت سرویس[22] و غیره را برآورد می‌کند. (نمونه‌هایی از چنین ایجادکننده مشخصات می‌تواند در [12] و [13] یافت شود). برآورد برای تمامی مدیران دستگاه‌ها که نمونه‌هایی از ماژول‌های متعلق به سایت را اجرا می‌کنند، ارسال می‌شود. به‌طور مشابه، ارسال-کننده درخواست، درخواست‌های کاربران را برای پردازش برای نمونه‌های ماژول‌های متعلق به سایت ارسال می‌کند. تصمیم درخواست ارسال، سیاست تخصیص منابع و محدودیت‌هایی از قبیل وابستگی جلسه[23] را به حساب می‌آورد. شکل 2 (سمت راست) مؤلفه های یک مدیر سایت و چگونگی ارتباط با مدیران دستگاه را نشان می‌دهد.



شکل 2. معماری برای واسط ابر (سمت چپ) و مؤلفه های برای رسیدگی درخواست‌ها و تخصیص منابع (سمت راست) [7].

[21] Profiler
[22] QoS
[23] Session affinity

از نقطه نظر مصرف انرژی، ما یک دستگاه که دارای دو حالت فعال و آماده‌به‌کار است را در نظر می‌گیریم. یک دستگاه فعال تمام لایه‌های نرم‌افزار و مؤلفه‌های نشان داده‌شده در شکل 2 را اجرا می‌کند و بنابراین سطح بالایی از انرژی را مصرف می‌کند، درحالی‌که یک دستگاه آماده‌به‌کار، همه مؤلفه‌های در شکل 2 را اجرا نمی‌کند و در نتیجه مصرف انرژی آن کم یا ناچیز است. در این کار به خاطر دلایل آورده شده در [14]، ما خودمان را محدود به حالت آماده‌به‌کار می‌کنیم، با دانستن اینکه استاندارد *ACPI* سطوح مختلف حالت آماده‌به‌کار را تعریف می‌کند [15]. حالت آماده‌به‌کار در کار ما، می‌تواند به‌عنوان حالت *ACPI G2* در مشخصات *ACPI* تحقق یابد. دلیل این است که حالت، فعال‌سازی یک دستگاه راه دور از طریق یک بسته شبکه[۲۴] را اجازه می‌دهد. هر دستگاه در ابر، با خدمات مخزن[۲۵] دستگاه نشان داده‌شده در شکل 3 ثبت می‌شود و ردیابی حالت انرژی دستگاه را نگه می‌دارد، به‌عنوان مثال، فعال یا آماده‌به‌کار.



شکل 3. خدمات مخزن دستگاه

مؤلفه مدیر منابع تعیین می‌کند که آیا یک دستگاه می‌تواند به حالت آماده‌به‌کار برود یا یک دستگاه اضافی برای فعال شدن نیاز است. در مورد اول، آن یک پیام تعویض به حالت آماده‌به‌کار، به خدمات مخزن دستگاه می‌فرستد که

[24] Wake-on-LAN packet
[25] Pool

متعاقباً دستگاه را به حالت آماده‌به‌کار تعویض می‌کند. در مورد دوم، آن پیام فعال کردن دستگاه را به خدمات می‌فرستد، که شناسه یک دستگاه فعال را اگر یکی در دسترس باشد، بر می‌گرداند. در ادامه این مقاله به قابلیت مؤلفه مدیر منابع تمرکز می‌شود. برای دیگر مؤلفه‌های معماری ما، مانند مدیر پوشش و ایجادکننده مشخصات تقاضا، ما به راه‌حل‌های شناخته شده تکیه می‌کنیم. یک طراحی برای خدمات مخزن دستگاه، بخشی از کارهای آینده ما است.

3. مدل‌سازی تخصیص منابع و راه‌حل عمومی ما

برای این کار، ما یک ابر که دارای منابع محاسباتی (مانند پردازنده) و منابع حافظه است را در نظر می‌گیریم که در دستگاه‌های زیرساخت ابر موجود است. ما دستگاه‌های موجود در ابر را همگن فرض می‌کنیم، به این معنا که پردازنده و ظرفیت حافظه آن‌ها و نیز خواص مصرف انرژیشان یکسان هستند.

ما بحث را به موردی که در آن تمامی دستگاه‌ها به یک دسته (خوشه) تک متعلق است محدود می‌کنیم، و به‌عنوان همیار[۲۶] در وظیفه تخصیص منابع همکاری می‌کنند. مسئله خاصی که ما به آن می‌پردازیم، قرار دادن ماژول‌ها (دقیق‌تر، نمونه‌های یکسان ماژول‌ها) در دستگاه‌ها و تخصیص منابع ابر به این ماژول‌ها، به‌طوری‌که اهداف ابر به دست آید می‌باشد.

ما مسئله مدیریت را به‌عنوان یک مسئله بهینه‌سازی که در آن راه‌حل یک ماتریس پیکربندی است که مؤلفه‌های زمان‌بند ماژول و ارسال‌کننده درخواست را کنترل می‌کند، مدل می‌کنیم. در زمان گسسته، رویدادهایی رخ می‌دهد، مانند تغییرات بار، افزودن و حذف سایت یا دستگاه و غیره. در پاسخ به چنین رویدادهایی، مسئله بهینه‌سازی به‌منظور حفظ پیکربندی بهینه، دوباره حل می‌شود. ما مدلمان برای تخصیص منابع را در بخش 3-الف معرفی می‌کنیم و الگوریتم عمومی برای مدیریت منابع را در بخش 3-ب ارائه می‌هیم.

[26] Peer

A. مدل

ما ابر را به‌عنوان یک سیستم با مجموعه‌ای از سایت ها (S) و مجموعه‌ای از دستگاه‌ها (N) که سایت‌ها را اداره می‌کنند، مدل می‌کنیم. هر سایت $s \in S$ از مجموعه‌ای از ماژول‌ها که با M_s نشان داده می‌شود، تشکیل شده است، و مجموعه‌ای از تمام ماژول‌های موجود در ابر، $M = \bigcup_{s \in S} M_s$ است.

ما تقاضا پردازنده را با بردار $\omega(t) = \left[\omega_1(t), \omega_2(t), \dots, \omega_{|M|}(t)\right]^T$ و تقاضا حافظه را با بردار $\gamma = \left[\gamma_1, \gamma_2, \dots, \gamma_{|M|}\right]^T$ مدل می‌کنیم، با این فرض که تقاضا پردازنده وابسته به زمان است، درحالی‌که تقاضا حافظه وابسته به زمان نیست [16].

ما یک سیستم را در نظر می‌گیریم که ممکن است بیش از یک نمونه از ماژول m که هر یک در دستگاه مختلف است را اداره کند. که در این صورت تقاضا پردازنده آن، در میان نمونه‌هایش تقسیم می‌شود. تقاضا $\omega_{n,m}(t)$ از یک نمونه m در حال اجرا در دستگاه n با $\omega_{n,m}(t) = \alpha_{n,m}(t)\omega_m(t)$ نشان داده می‌شود که در آن $\sum_{n \in N} \alpha_{n,m}(t) = 1$ و $\alpha_{n,m}(t) \geq 0$ است. ما ماتریس A با عناصر $\alpha_{n,m}(t)$ را پیکربندی (ماتریس) سیستم می‌نامیم. A یک ماتریس غیر منفی با $1^T A = 1^T$ است.

یک دستگاه $n \in N$ در ابر، دارای ظرفیت پردازنده Ω و ظرفیت حافظه Γ است. ما Ω و Γ را برای نشان دادن بردارهای ظرفیت پردازنده و حافظه تمامی دستگاه‌ها در سیستم استفاده می‌کنیم. یک نمونه m در حال اجرا در دستگاه n، $\omega_{n,m}(t)$ منبع پردازنده و γ_m منبع حافظه از n را تقاضا می‌کند. دستگاه n برای ماژول m ظرفیت پردازنده $\widehat{\omega}_{n,m}(t)$ (که ممکن است با $\omega_{n,m}(t)$ متفاوت باشد) و ظرفیت حافظه γ_m را تخصیص می‌دهد.

مقدار برای $\widehat{\omega}_{n,m}(t)$ به سیاست تخصیص $\widehat{\Omega}(t)$ در ابر بستگی دارد. سیاست خاصی که ما در این استفاده کردیم، تخصیص $\widehat{\omega}_{n,m}(t) = \frac{\omega_{n,m}(t)}{\sum_i \omega_{n,i}} \Omega$ است.

B. GRMP: پروتکل مدیریت منابع عمومی[۲۷]

با توجه به مدل بالا، ماتریس پیکربندی A تعیین می‌کند که چگونه منابع ابر به سایت‌ها تخصیص داده می‌شود. ما طرفدار استفاده از پروتکل *gossip* برای محاسبه کارآمد این ماتریس برای ابر مقیاس بزرگ هستیم. پروتکل‌های *gossip* از پروتکل‌های مبتنی بر دور[۲۸] هستند که در آن، در هر دور، یک گره[۲۹] یک زیرمجموعه از گره‌های دیگر را برای تعامل انتخاب می‌کند. انتخاب گره اغلب به‌صورت احتمالی است و به‌عنوان گره‌هایی که با دورهای بیشتر عمل می‌کند، حالت آن‌ها به یک حالت مطلوب همگرا می‌شود. پروتکل *gossip* برای تعدادی وظایف مدیریت شامل انتشار اطلاعات با یک روش قوی، محاسبه تراکم[۳۰]، و نیز ایجاد و حفظ پوشش ارائه شده است.

در این زیر بخش، ما یک پروتکل *gossip* عمومی برای تخصیص منبع معرفی می‌کنیم که می‌تواند برای اهداف مدیریت مختلف با نمونه معرفی شود. ما این را پروتکل *GRMP* (پروتکل مدیریت منابع عمومی) می‌نامیم. *GRMP* در مؤلفه مدیر منبع تمام دستگاه‌ها موجود در ابر اجرا می‌شود. (شکل 2 را ببینید). مجموعه‌ای از دستگاه-های منتخب برای تعامل، توسط مؤلفه مدیر پوشش از مدیر دستگاه حفظ می‌شود.

GRMP در زمان گسسته استناد شده است. بسته به استقرار خاص، فراخوانی[۳۱] ممکن است دوره‌ای، در پاسخ به یک رویداد (مانند تغییر قابل توجه بار یا اضافه شدن دستگاه جدید)، یا ترکیبی از هر دو باشد. در طی هر فراخوانی *GRMP*، هر دستگاه r_{max} دور را انجام می‌دهد و ماتریس پیکربندی A را تولید می‌کند. مقدار برای r_{max} بستگی به نمونه خاص *GRMP* دارد. ماتریس A در سراسر دستگاه‌های سیستم توزیع می‌شود و شروع و توقف نمونه‌های ماژول را کنترل می‌کند و سیاست‌های کنترل برای زمان‌بند ماژول و ارسال‌کننده درخواست را تعیین می‌کند. مؤلفه مدیر منبع تعیین می‌کند که آیا ماتریس پیکربندی محاسبه‌شده، اجرا شده است یا نه. ما فرض می‌کنیم که زمان آن برای *GRMP* به‌منظور محاسبه یک پیکربندی جدید، A، در مقایسه با زمان بین رویدادهایی که باعث اجرای متوالی پروتکل می‌شود، کم است. در زمان مقداردهی اولیه، *GRMP* یک پیکربندی امکان‌پذیر سیستم را به‌عنوان

[27] Generic Resource Management Protocol
[28] Round-based
[29] Node
[30] Aggregates
[31] Invocation

ورودی می‌خواند که می‌تواند با استفاده از [7] و [17] محاسبه شود. در فراخوانی بعد، پروتکل، پیکربندی ماتریس تولیدشده در طی اجرای قبلی را به‌عنوان ورودی می‌خواند.

شبه کد *GRMP* در الگوریتم 1 مشخص شده است. پروتکل، به اصطلاح الگوی تعامل فشار-کشش[32] *gossip* را پیروی می‌کند که ما با رشته[33] فعال یا غیر فعال[34] در هر دستگاه پیاده‌سازی می‌کنیم. برای حفظ سادگی ارائه، همزمانی رشته را حذف می‌کنیم که از به روز رسانی متقارن حالت محلی، با رشته‌های فعال و غیر فعال جلوگیری می‌کند.

الگوریتم 1. پروتکل ***GRMP*** ماتریس پیکربندی $\boldsymbol{A}$ را محاسبه می‌کند. کد برای دستگاه $\boldsymbol{n}$ است.

initialization
1: read $\omega, \gamma, \Omega, \Gamma, row_n(A)$;
2: $initInstance()$;
3: start passive and active threads;

active thread
1: **for** $r = 1$ to r_{max} **do**
2: $n' = choosePeer()$;
3: $send(n', row_n(A))$; $row_{n'}(A) = receive(n')$;
4: $updatePlacement(n', row_{n'}(A))$;
5: sleep until end of round;
6: write $row_n(A)$;

passive thread
1: **while** *true* **do**
2: $row_{n'}(A) = receive(n')$; $send(n', row_n(A))$;
3: $updatePlacement(n', row_{n'}(A))$;

GRMP یک پروتکل عمومی است، به این معنا که سه روش انتزاعی (غیر عملی) باید به‌منظور محاسبه یک ماتریس پیکربندی برای یک هدف مدیریت منابع خاص اجرا شود.

1) *initInstance*() روش مقداردهی اولیه برای پروتکل *gossip* خاص است.

2) *choosePeer*() روشی برای انتخاب یک همیار برای تعامل *gossip* است.

3) *updatePlacement*() روشی برای محاسبه دوباره حالت محلی در طی یک تعامل *gossip* است.

[32] Push-pull
[33] Thread
[34] Passive

در زیر بخش 4-ب، ما یک نمونه‌ای از *GRMP* که *GRMP-Q* نامیده می‌شود را ارائه می‌دهیم، که تخصیص منابع را باهدف کاهش مصرف انرژی انجام می‌دهد. پروتکل *gossip* که ما در کار قبلیمان توسعه دادیم، می‌تواند به‌عنوان یک نمونه از *GRMP* تفسیر شود [7]. آن پروتکل هدف تخصیص عادلانه منابع پردازنده به سایت‌ها را انجام می‌دهد. درحالی‌که روش‌های *initInstance()* و *choosePeer()* در یک روش مشابه مانند اونهایی که برای *GRMP-Q* بود اجرا می‌شوند، معنی *updatePlacement()* متفاوت است. این، حالت‌های محلی دستگاه‌های در حال تعامل را در چنین راهی که درخواست‌های نسبی پردازنده به روز رسانی می‌کند، به‌صورت $\frac{\sum_m \omega_{n,m}}{\Omega}$ برای n دستگاه محاسبه می‌شود، برابر هستند. این پروتکل تحت شرایط خاصی بهینه است، به این معنی که پروتکل دنباله‌ای از پیکربندی‌ها تولید می‌کند، هنگامی‌که دورهای اجرا، به حالت بهینه به‌صورت نمایی تند همگرا می‌شود.

4. مسئله و راه‌حل ما

A. مدیریت منابع به‌عنوان یک مسئله بهینه‌سازی

هدف اول، برآوردن درخواست کاربر است، اگر با منابع خوشه در دسترس ممکن است (برای مثال، زیر بار) و تخصیص عادلانه منابع اگر با منابع خوشه در دسترس ممکن نیست (برای مثال، سربار). ما این را با استفاده از مفهوم سودمندی[35] رسمی می‌کنیم. ما سودمندی تولیدشده توسط یک نمونه از ماژول m روی دستگاه n را به‌عنوان نسبت ظرفیت پردازنده تخصیص‌یافته به درخواست نمونه در آن دستگاه خاص تعریف می‌کنیم. یعنی $u_{n,m}(t) = \frac{\hat{\omega}_{n,m}(t)}{\omega_{n,m}(t)}$. (نمونه‌ای با $\omega_{n,m} = 0$، سودمندی بی‌نهایت تولید می‌کند). سودمندی تولیدشده توسط یک سایت به‌صورت $U^c(t) = u(s,t) = min_{n,m\epsilon M_s} u_{n,m}(t)$ تعریف می‌شود. سودمندی ابر به‌صورت $min_{s|u(s,t)\leq 1} u(s,t) = min_{n,m|u_{n,m}\leq 1} u_{n,m}(t)$ تعریف می‌شود. هدف اول می‌تواند به‌عنوان حداکثرسازی $U^c(t)$ بیان شود، که تضمین می‌کند که تمام درخواست‌های سایت، در صورت زیر بار، برآورده می-

[35] Utility

شود. در صورت سربار، حداکثرسازی $U^c(t)$، حداکثر-حداقل عادلانه[۳۶] در مورد تخصیص منابع پردازنده برای سایت‌ها را تضمین می‌کند.

هدف دوم، حداقل سازی مصرف انرژی ابر است. ما مصرف انرژی دستگاه n را با تابع زیر مدل می‌کنیم.

$$P_n(t) = \begin{cases} 0 & if\ row_n(A)(t)1 = 0 \\ 1 & otherwise \end{cases}$$

$P_n(t) = 0$ به این معنی است که دستگاه می‌تواند به حالت آماده‌به‌کار تعویض شود، و $P_n(t) = 1$ به این معنی است که دستگاه باید فعال بماند. ما مصرف انرژی ابر را با $P^c(t) = \sum_n P_n(t)$ بیان می‌کنیم. بنابراین هدف دوم حداقل سازی $P^c(t)$ است.

مسئله تخصیص منابع تطبیق پیکربندی $A(t)$ به پیکربندی جدید $A(t+1)$ است، به‌طوری‌که هدف سیستم مدیریت منابع برای تقاضا جدید $\omega(t+1)$ به دست آید. هدف سوم شناسایی یک پیکربندی است که تابع هزینه $c^*(A(t), A(t+1))$ را حداقل کند. این تابع هزینه جریمه مرتبط با تغییر پیکربندی $A(t)$ به $A(t+1)$ را حساب می‌کند. چنین جریمه‌ای ممکن است منعکس‌کننده باشد. برای مثال، سطح بالایی از مصرف پهنای باند شبکه یا یک‌زمان وقفه سرویس طولانی در طول پیکربندی دوباره. (تابع هزینه‌ای که ما در این کار در نظر می‌گیریم، تعداد نمونه‌هایی از ماژول‌ها را می‌شمارد که برای پیکربندی مجدد سیستم از پیکربندی فعلی به پیکربندی جدید شروع می‌شوند).

حالا ما مسئله بهینه‌سازی را با استفاده از سه هدف بحث شده در بالا رسمی می‌کنیم. یک ابر با ظرفیت پردازنده Ω و ظرفیت حافظه Γ را در نظر بگیرید. بنابراین با توجه به پیکربندی $A(t)$، بردار تقاضا پردازنده $\omega(t+1)$ و بردار تقاضا حافظه γ، مسئله، یافتن یک پیکربندی $A(t+1)$ است که مسائل بهینه‌سازی زیر را حل کند.

$$maximize\ U^c(t+1)$$

$$\underset{(OP)}{\text{[illegible]}ize\ P^c(t+1)}$$

$$minimize\ c^*(A(t), A(t+1))$$

به قسمی که: $A(t+1) \geq 0, 1^T A(t+1) = 1^T$

[36] Max-min fairness

$$\hat{\Omega}\big(A(t+1),\omega(t+1)\big)1 \preccurlyeq \Omega$$
$$sign\big(A(t+1)\big)\gamma \preccurlyeq \Gamma.$$

این مسئله بهینه‌سازی اهداف را اولویت‌بندی کرده است. این به این معنی است که، بین تمام پیکربندی‌های A که سودمندی ابر U^c را حداکثر می‌کند، ما آن پیکربندی‌هایی که مصرف انرژی P^c را حداقل می‌کند انتخاب می‌کنیم. خارج این پیکربندی‌ها، ما آن را که تابع هزینه c^* را حداقل می‌کند انتخاب می‌کنیم. محدودیت‌های (OP) مربوط به (1) تقسیم تقاضا پردازنده هر ماژول به تقاضا از نمونه‌های ماژول، و (2) تضمین اینکه منابع پردازنده و حافظه تخصیص‌یافته در هر دستگاه نمی‌تواند بزرگ‌تر از ظرفیت در دسترس آن باشد.

اجازه دهید به‌طور خلاصه در مورد سختی (OP) توضیح دهیم. تقاضا حافظه برای یک ماژول قابل‌تقسیم نیست، به این معنی که تقاضا حافظه یک ماژول نمی‌تواند در بین نمونه‌هایش که در دستگاه‌های مختلف اجرا می‌شوند، تقسیم شود. این باعث NP-سخت[۳۷] شدن (OP) می‌شود. بااین‌حال، در بسیاری موارد عملی که در آن‌ها تقاضا حافظه ترکیب‌شده، به‌طور قابل‌توجهی کوچک‌تر از ظرفیت حافظه ابر است، یک راه‌حل برای (OP) به‌راحتی می‌تواند یافت شود.

B. راه‌حل *GRMP-Q* ما: یک راه‌حل ابتکاری برای (OP)

به‌عنوان یک نمونه از *GRMP*، *GRMP-Q* سه روش انتزاعی *GRMP*، نشان داده‌شده در الگوریتم 2 را پیاده‌سازی می‌کند. در روش *initInstance*، دستگاه n مقداردهی اولیه می‌کند N_n را، که N_n مجموعه‌ای از دستگاه‌ها است که ماژول‌های مشترک با n را اجرا می‌کنند. یک دستگاه n ترجیح می‌دهد به اجرای مرحله *gossip* با دیگر دستگاه‌های $j \in N_n$. دلیل این است که، بار می‌تواند بین دو دستگاه بدون نیاز به حافظه اضافی و بدون هیچ هزینه پیکربندی دوباره، حرکت کند. بااین‌حال، همیشه انتخاب j از N_n، ممکن است در ابر منجر به تقسیم شدن در مجموعه‌های متلاشی دستگاه‌های در حال تعامل شود. برای جلوگیری از این وضعیت، n گاهی اوقات با یک دستگاه خارج از مجموعه N_n جفت می‌شود. تابع انتخاب همسایه *choosePeer()*، این حالت را به‌صورت زیر پیاده‌سازی

[37] NP-hard

می‌کند: آن، یک دستگاه که به‌صورت یکنواخت تصادفی از مجموعه N_n با (قابل پیکربندی) احتمال p و از مجموعه $N\text{-}N_n$ با احتمال $1\text{-}p$ را برمی‌گرداند.

هسته پروتکل در تابع *updatePlacement* پیاده‌سازی می‌شود که نمونه‌های ماژول را از یک دستگاه به دستگاه دیگر حرکت می‌دهد. هدف از این حرکت، توسط تقاضا پردازنده نسبی دستگاه‌های شرکت‌کننده مشخص می‌شود که برای دستگاه n به‌عنوان $v_n = \sum_m \omega_{n,m}/\Omega$ تعریف می‌شود. به‌طور خاص، برای دستگاه‌های n و j، اگر $v_n + v_j \geq 2$، پروتکل تخمین می‌زند که ابر در حالت سربار است و یک تابع باهدف دستیابی به عدالت برای منابع پردازنده فراخوانده می‌شود. این تابع (که در [7] شرح داده‌شده) ماژول‌ها را از دستگاه با تقاضای نسبی بالاتر به دستگاه با تقاضای نسبی پایین‌تر، باهدف تساوی v_n و v_j حرکت می‌دهد.

اگر $v_n + v_j < 2$، پروتکل تخمین می‌زند که ابر در حالت زیر بار است و توابع باهدف کاهش مصرف انرژی ابر فراخوانده می‌شوند، درحالی‌که تقاضای سایت‌ها برآورده شود. این توابع، *packNonShared* که همیشه فراخوانده می‌شود، و *packshared* که تنها اگر دو دستگاه ماژول‌ها را به اشتراک بگذارند فراخوانده می‌شود، هستند. توابع بر مبنای دو مفهوم زیر هستند.

مفهوم اول، که با تابع *pickSrcDest* پیاده‌سازی می‌شود، تضمین می‌کند که پروتکل در درجه اول ماژول‌ها را از دستگاه دارای سربار، به دستگاه دارای زیر بار حرکت می‌دهد، باهدف برآوردن تقاضا نمونه‌های ماژول بر روی دستگاه دارای سربار. از سوی دیگر، اگر هر دو دستگاه دارای زیر بار هستند، پروتکل ماژول‌ها را از دستگاه با بار کمتر به دستگاه با بار بالاتر حرکت می‌دهد، در تلاش برای بسته کردن[38] کامل یک دستگاه یا آزاد کردن[39] دستگاه دیگر.

مفهوم دوم مربوط به بهره‌وری بسته‌بندی پروتکل است. به‌طور خاص، آن تلاش می‌کند برای جلوگیری از شرایطی که در آن یک نوع تک از منبع (مانند فقط پردازنده یا حافظه) یک دستگاه استفاده می‌شود، درحالی‌که دیگری استفاده نمی‌شود. چنین شرایطی بهره‌وری بسته‌بندی پروتکل را کاهش می‌دهد و ازاین‌رو باعث کاهش مصرف انرژی می‌شود. بنابراین، در طول تعامل، پروتکل منابع غالب در مقصد را شناسایی می‌کند (مانند نوع منبعی که دارای

[38] Pack
[39] Freeing up

تقاضای نسبی بزرگ‌تر است)، و ماژول‌ها در دستگاه منبع را انتخاب می‌کند، به‌طوری‌که آن‌ها منابع غالب کمتر دارند. (در شبه کد، تقاضای حافظه نسبی به‌عنوان $g_n = \sum_m \gamma_m / \Gamma$ تعریف می‌شود). توصیف کامل تابع در [18] موجود است.

الگوریتم 2. پروتکل **GRMP-Q**. یک نمونه از **GRMP** برای حل (**OP**). کد برای دستگاه $\boldsymbol{n}$ است.

```
initInstance()
1: read N_n;

choosePeer()
1: if rand(0..1) < p then
2:    return unifrand(N_n);
3: else
4:    return unifrand(N − N_n);

updatePlacement(j, row_j(A))
1: if (v_n + v_j ≥ 2) then
2:    equalize(j, row_j(A));
3: else
4:    if j ∈ N_n then
5:       packShared(j);
6:    packNonShared(j);

packShared(j)
1: (s, d) = pickSrcDest(j); Δω_d = Ω − Σ_m ω_{d,m};
2: if v_s > 1 then Δω_s = Σ_m ω_{s,m} − Ω; else Δω_s
   = Σ_m ω_{s,m};
3: Let mod be the list of modules shared by
   s and d, sorted by decreasing γ_{s,m}/ω_{s,m};
4: while mod ≠ ∅ ∧ Δω_s > 0 ∧ Δω_d > 0 do
5:    m = remove first element from mod;
6:    δω = min(Δω_d, Δω_s, ω_{s,m}); Δω_d −=δω;
7:    Δω_s −=δω; δα=α_{s,m} δω/ω_{s,m}; α_{d,m} +=δα; α_{s,m} −=δα;

packNonShared(j)
1: (s, d) = pickSrcDest(j);
2: Δγ_d = Γ − Σ_m γ_{d,m}; Δω_d = Ω − Σ_m ω_{d,m};
3: if v_s > 1 then Δω_s = Σ_m ω_{s,m} − Ω; else Δω_s =
   Σ_m ω_{s,m};
4: if v_d ≥ g_d then sortCri = γ_{s,m}/ω_{s,m}; else
   sortCri = ω_{s,m}/γ_{s,m};
5: Let mod be the list of modules on s
   not shared with d, sorted by decreasing
   sortCri;
6: while mod ≠ ∅∧Δγ_d > 0∧Δω_d > 0∧Δω_s > 0 do
7:    m = remove first element from mod;
8:    δω = min(Δω_s, Δω_d, ω_{s,m}); δγ = γ_{s,m};
9:    if Δγ_d ≥ δγ then
10:      δα = α_{s,m} δω/ω_{s,m}; α_{d,m} +=δα; α_{s,m} −=δα;
11:      Δγ_d −= δγ; Δω_d −= δω; Δω_s −= δω;

pickSrcDest(j)
1: dest=arg max(v_n, v_j); src=arg min(v_n, v_j);
2: if v_dest > 1 then swap dest and src;
3: return (src, dest);
```

C. خصوصیات GRMP-Q

ازآنجاکه $GRMP\text{-}Q$ یک راه‌حل ابتکاری است، پیکربندی که آن تولید می‌کند، به‌طورکلی برای (OP) بهینه نیست. برای درک خصوصیات پروتکل، ما مفاهیم مفید را معرفی می‌کنیم: ضریب بار پردازنده[40] $CLF = \frac{\omega^T 1}{|N|\Omega}$ و ضریب بار حافظه[41] $MLF = \frac{\gamma^T 1}{|N|\Gamma}$. ابر در حالت سربار است، هرگاه $CLF > 1$. به این معنا که تقاضای کل برای منابع پردازنده بیش از ظرفیت موجود در ابر است. (این مقاله مورد $MLF > 1$ را شامل نمی‌شود، زیرا مکان اولیه برای چنین بار در ابر ممکن نیست و تقاضاهای حافظه ثابت فرض شده است).

الف) ابر در حالت سربار ($CLF > 1$ و $MLF < 1$): پروتکل در یک راهی که تمام دستگاه‌های موجود در ابر، درنهایت سربار می‌شوند. هنگامی‌که این مورد است، پروتکل به همان صورت پروتکل منصفانه شرح داده‌شده در [7] عمل می‌کند، به این معنی که آن برای تخصیص منبع پردازنده در سراسر سایت‌ها با استفاده از سیاست منصفانه حداکثر-حداقل تلاش می‌کند.

ب) تقاضا حافظه بسیار کمتر از ظرفیت ($MLF \ll 1$): بعد از هر تعامل $gossip$، دستگاه‌های در حال تعامل در یکی از حالت‌های زیر هستند: (1) هر دو دستگاه دارای بار برابر هستند. (2) یک دستگاه حداکثر بار پردازنده حمل می‌کند. (3) یک دستگاه هیچ باری حمل نمی‌کند. تحت این شرایط، پیکربندی محاسبه‌شده توسط پروتکل، به یک راه‌حل بهینه از (OP) همگرا می‌شود—اگر ما هزینه پیکربندی دوباره را نادیده بگیریم. اگر $CLF < 1$، راه‌حل بهینه اشاره دارد به اینکه، $\lfloor |N|CLF \rfloor$ تعداد دستگاه حداکثر بار را حمل می‌کنند و $|N| - \lfloor |N|CLF \rfloor$ تعداد دستگاه هیچ باری را حمل نمی‌کنند، درحالی‌که تمام تقاضای سایت‌ها برآورده می‌شود.

ج) حالت کلی ($CLF < 1$ و $MLF < 1$): با طراحی، پروتکل اولویت را به حرکت بار، دور از دستگاه سربار شده روی بار در حال انتقال، به‌منظور کاهش مصرف انرژی می‌دهد. به‌عنوان نتیجه، ما می‌توانیم بگوییم که اگر پیکربندی

[40] CPU load factor
[41] Memory load factor

جدید، پروتکل دستگاه‌هایی که بار حمل نمی‌کنند را تولید کند، دستگاه‌های با بار، به‌طور کامل تقاضا را برآورده می‌کنند.

5. ارزیابی از طریق شبیه‌سازی

ما *GRMP-Q* را از طریق شبیه‌سازی گسترده با استفاده از شبیه‌ساز رویداد گسسته شبیه‌سازی کرده‌ایم که ما در سازمان[42] گسترش دادیم. ما یک سیستم توزیع‌شده را شبیه‌سازی می‌کنیم که مؤلفه‌های مدیر دستگاه تمام دستگاه‌های موجود در ابر را اجرا می‌کند. به‌طور خاص، این مدیرهای دستگاه‌ها، پروتکل *GRMP-Q* را اجرا می‌کنند، که تخصیص ماتریس A را محاسبه می‌کند، و همچنین پروتکل *CYCLON*، که تابع انتخاب یک همسایه تصادفی را برای *GRMP-Q* فراهم می‌کند. شبیه‌ساز همچنین الگوریتم شرح داده‌شده در [7] برای محاسبه پیکربندی امکان‌پذیر اولیه ابر را پیاده‌سازی می‌کند. رویدادهای خارجی برای این شبیه‌سازی، تغییرات در بردار تقاضای ω هستند.

معیارهای ارزیابی: ما کاهش مصرف انرژی $\frac{|N|-P^c}{|N|}$ را به‌عنوان کسری از دستگاه‌های موجود در ابر اندازه می‌گیریم که توسط پروتکل آزاد می‌شوند. دوم، ما عدالت تخصیص منابع را از طریق ضریب تغییرات سودمندی سایت، که به‌عنوان نسبت انحراف معیار به متوسط سودمندی‌ها محاسبه می‌شود، اندازه می‌گیریم. سوم، ما تقاضای برآورده شده را به‌عنوان کسری از سایت‌ها که سودمندی بزرگ‌تر یا مساوی 1 تولید می‌کند، اندازه می‌گیریم. درنهایت، ما هزینه پیکربندی دوباره را به‌عنوان نسبت نمونه‌های ماژول شروع‌شده به نمونه‌های ماژول در حال اجرا، در هر دستگاه، اندازه می‌گیریم.

تولید بردار تقاضای ω و γ: تعدادی از ماژول‌های یک سایت، با استفاده از توزیع گسسته پواسون با میانگین 1، افزایش‌یافته با 1، انتخاب می‌شود. تقاضای حافظه یک ماژول، به‌طور یکنواخت تصادفی از مجموعه c_γ انتخاب می‌شود. {128 مگابایت، 256 مگابایت، 512 مگابایت، 1 گیگابایت، 2 گیگابایت}. برای سایت s، در هر تغییر در تقاضا، ایجادکننده مشخصات تقاضا، تقاضاهای پردازنده انتخاب‌شده از توزیع نمایی با میانگین $\omega(s)$ را تولید می‌کند. ما

[42] In-house

توزیع را برای $\omega(s)$، در میان تمام سایت‌ها، برای توزیع زیپف[43] شدن با $\alpha = 0.7$ انتخاب می‌کنیم. اثبات در [19] موجود است. حداکثر مقدار برای توزیع، c_ω است. 500 گیگ واحد پردازنده و اندازه جمعیت استفاده‌شده، 20000 است. برای ماژول m از سایت s، ما فاکتور تقاضا β_m با $\sum_{m\epsilon M_s} \beta_m = 1$، انتخاب‌شده به‌صورت یکنواخت تصادفی را انتخاب می‌کنیم، که سهم ماژول m در تقاضای سایت s را توصیف می‌کند. c_γ و c_ω فاکتورهای مقیاس هستند (پایین را ببینید).

پارامترهای سناریو: ما عملکرد تخصیص منابع پروتکل *GRMP-Q* را تحت نمونه‌های متفاوت بار پردازنده و حافظه، ارزیابی می‌کنیم، که ما با تغییر c_γ و c_ω دگرگون می‌کنیم. تمام دستگاه‌های موجود در ابر دارای ظرفیت پردازنده یکسان 34.513 گیگ واحد پردازنده، و ظرفیت حافظه 36.409 گیگابایت هستند. این مقادیر، *MLF=CLF=0.5* برای $c_\gamma = c_\omega = 5$ را می‌دهد. ما پارامترهای زیر را استفاده می‌کنیم، مگر اینکه غیر این‌ها ذکرشده باشد:

- *|N|=10000*، *|S|=24000*، r_{max}*=30*، $p = \frac{|N_n|}{1+|N_n|}$
- حداکثر تعداد نمونه‌ها/ماژول: 100 و تعداد تغییرات بار در طول اجرا: 100

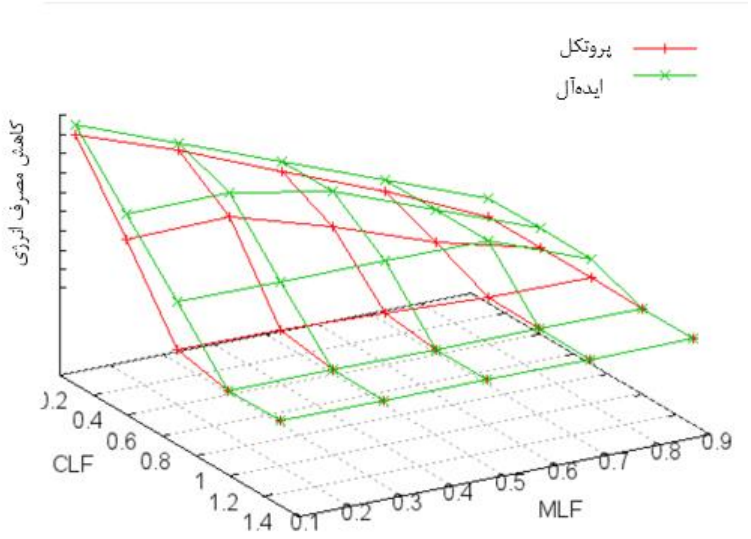
A. عملکرد GRMP-Q تحت CLF و MLF مختلف

در این سناریو، ما عملکرد *GRMP-Q* را برای *CLF={0.1,0.4,0.7,1,1.3}* و *MLF={0.1,0.3,0.5,0.7,0.9}*، توسط اندازه‌گیری معیارهای لیست‌شده در بالا، ارزیابی می‌کنیم. ما نتایجمان را با یک سیستم ایده آل که دارای تراکم ظرفیت حافظه و پردازنده ابر است و اینکه انرژی را طبق تابع $P_{lb}^C = [min\ (1, max(CLF, MLF))]$ مصرف می‌کند، مقایسه می‌کنیم. (P_{lb}^C یک کران پایین برای P^C است که یک تخمین خوب از مقدار بهینه برای P^C، مقدار پایین MLF است). در این مقاله، نتایج مربوط به کاهش انرژی و تقاضای برآورده شده را گزارش می‌کنیم. ارزیابی کامل پروتکل، در [18] موجود است.

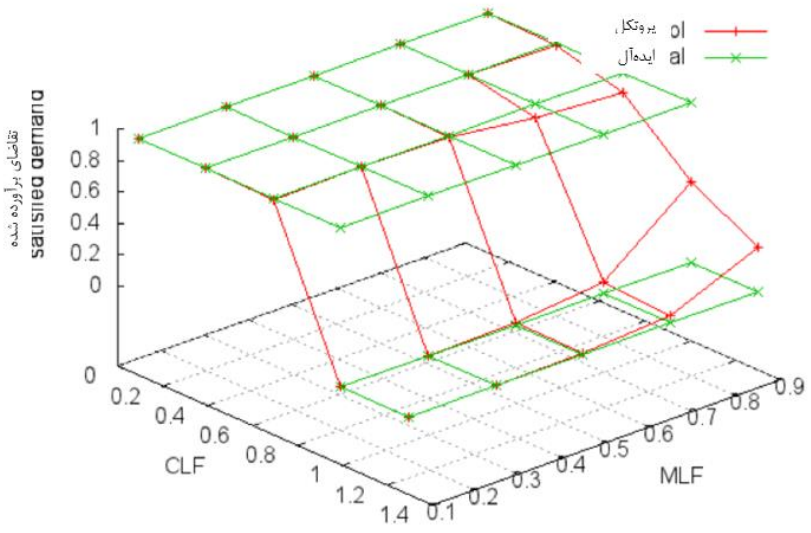
[43] Zipf

1) کاهش مصرف انرژی: شکل 4الف، کاهش در مصرف انرژی به‌دست‌آمده توسط *GRMP-Q* برای مقادیر مختلف *CLF* و *MLF* را نشان می‌دهد. همان‌طور که انتظار می‌رفت، این مقدار برای افزایش *CLF* و *MLF*، کاهش می‌یابد. برای نمونه، کاهش در مصرف انرژی، از 85٪ برای $CLF=MLF=0.1$ به 0 برای $CLF \geq 1$ و $MLF \geq 0.9$، را کاهش می‌دهد. این ازآنجا مورد انتظار است که تعداد دستگاه‌های موردنیاز برای اجرا و برآورده کردن تقاضاهای همه سایت‌ها، با هر دو *CLF* و *MLF* افزایش می‌یابد. این کاهش همچنین به 0 کاهش می‌یابد برای $CLF \geq 1$.

2) تقاضای برآورده شده: شکل 4ب نشان می‌دهد که تقاضای برآورده شده به هر دو *CLF* و *MLF* بستگی دارد. برای سیستم ایده آل، تقاضای برآورده شده فقط به *CLF* بستگی دارد. به‌طور خاص، تقاضای همه سایت‌ها، وقتی *CLF* کمتر از 1 است، برآورده می‌شود، و در غیر این صورت برآورده نمی‌شود. پروتکل ما بیشتر از 99٪ تقاضاهای سایت را در سناریو زیر بار برآورده می‌کند، به‌جز مورد *(CLF,MLF)=(0.7,0.7)* و *(CLF,MLF)=(0.7,0.9)*. همان‌طور که دیده می‌شود، برای مقادیر *CLF* بزرگ‌تر از 1، پروتکل ما یک تقاضای برآورده شده بزرگ‌تر از سیستم ایده آل، در مصرف ناعادلانه تخصیص پردازنده، به دست می‌آورد.



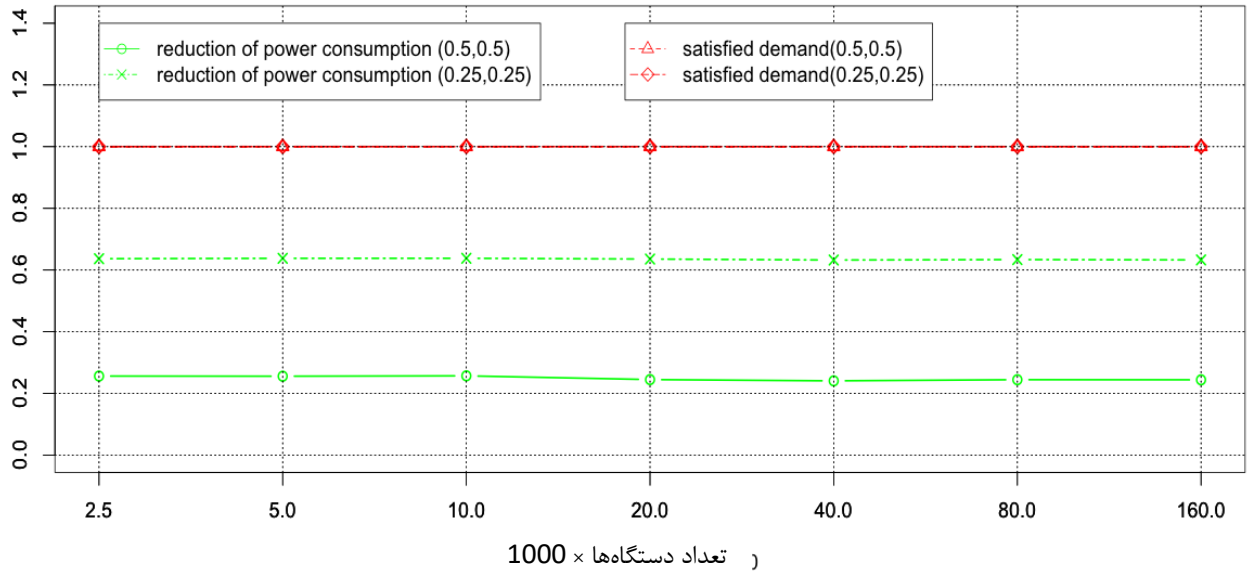
الف. کسری از دستگاه‌ها که می‌توانند در حالت آماده‌به‌کار قرار داده شوند.



ب. کسری از سایت‌ها با تقاضای برآورده شده

شکل 4. عملکرد پروتکل تخصیص منبع *GRMP-Q*، در تابعی از ضریب بار پردازنده (*CLF*) و ضریب بار حافظه (*MLF*) از ابر (10000 دستگاه و 24000 سایت)

در این سناریو، ما وابستگی معیارهای ارزیابی خود را روی اندازه ابر، اندازه می‌گیریم. برای رسیدن به این، ما شبیه-سازی را برای ابر با (2500، 5000، 10000، 20000، 40000، 160000) دستگاه و (6000، 12000، 24000، 48000، 96000، 384000) سایت انجام می‌دهیم. (حفظ نسبت سایت‌ها به دستگاه‌ها در 2.4). در تنظیمات، ما دو مجموعه متفاوت از *CLF* و *MLF* که $\{(0.25,0.25),(0.5,0.5)\}$ هستند را ارزیابی می‌کنیم. شکل 5 نتایج به‌دست‌آمده را نشان می‌دهد، که نشان می‌دهد که تمام معیارهای در نظر گرفته‌شده، وابسته به‌اندازه سیستم هستند. به‌عبارت‌دیگر، اگر تعداد دستگاه‌ها با نرخ یکسان از تعداد سایت‌ها رشد کند، (درحالی‌که ظرفیت‌های پردازنده و حافظه یک دستگاه، و همچنین تمام پارامترهای مشخص‌کننده یک سایت، مانند تقاضا، تعداد ماژول‌ها و غیره، یکسان بماند) ما انتظار داریم که تمام معیارهای در نظر گرفته‌شده ثابت باقی بماند. توجه داشته باشید که نتیجه-گیری ما به‌طور انحصاری به مقیاس‌پذیری پروتکل *GRMP-Q* مربوط است. سیستم مدیریت منابع کامل، شامل بسیاری توابع بیشتر است که در اینجا مورد ارزیابی قرار نگرفته است، برای نمونه، مقیاس‌پذیری انتخاب مؤثر یک همیار تصادفی.



شکل 5. مقیاس‌پذیری با توجه به تعداد دستگاه‌ها و ماشین‌ها.

6. کارهای مشابه

مسئله کاهش مصرف انرژی یک مرکز داده تحت محدودیت‌های عملکرد، به‌طور گسترده موردمطالعه قرارگرفته است [5]، [20]-[27] و همچنین تولید راه‌حل‌هایی که یک راه‌حل را برای چنین مسئله‌ای ترکیب می‌کند وجود دارد [4]. فاکتور کلیدی تفاوت کار ما، در مقایسه با تمام‌کارهای دیگر، استفاده از یک الگوریتم غیرمتمرکز برای محاسبه سیاست‌های تخصیص منبع در ابر است. این، در تضاد تند با راه‌حل‌های موجود، اجازه می‌دهد تا سیستم مدیریت منبع ما به 100000 دستگاه مقیاس کند، و به‌صورت پویا با تغییرات در تقاضای سایت‌های در حال اجرا انطباق یابد. ارائه کامل کارهای مشابه، در [18] موجود است.

7. بحث و نتیجه‌گیری

ما سه همکاری را با این مقاله ایجاد کردیدم. اول، ما مسئله حداقل سازی مصرف انرژی را از طریق تحکیم سرور، وقتی‌که سیستم در حالت زیر بار است و تخصیص منبع عادلانه در حالت سربار، معرفی و رسمی کردیم. دوم، ما *GRMP* را ارائه کردیم، که یک پروتکل *gossip* عمومی برای مدیریت منبع است که می‌تواند برای اهداف مختلف

معرفی شود. (یک پروتکل برای تخصیص منابع عادلانه از کار قبلی ما، درواقع یک نمونه از این پروتکل است). درنهایت، ما یک نمونه از *GRMP* را ارائه دادیم که یک راه‌حل ابتکاری برای مسئله حداقل سازی مصرف انرژی فراهم می‌کند، که ما برای مؤثر شدن و مقیاس‌پذیر شدن نشان دادیم.

مطالعات شبیه‌سازی *GRMP-Q* نشان می‌دهد که پروتکل طبق اهداف طراحی‌اش که در بخش 1 گفته شد، برای محدوده پارامترهای بررسی‌شده، انجام می‌دهد. برای نمونه، برای سناریو حالت زیر بار با *CLF=MLF=0.1*، پروتکل یک پیکربندی محاسبه می‌کند که در آن کمتر از 20٪ دستگاه‌ها، بار حمل می‌کنند، درحالی‌که تقاضای کاربر هنوز برآورده می‌شود. در سناریو حالت سربار، پروتکل منابع را به‌صورت عادلانه به سایت‌ها تخصیص می‌دهد، تا زمانی که حافظه کافی موجود است [18]. علاوه بر این، نتایج نشان می‌دهند که پروتکل مقیاس‌پذیر است، به این معنا که معیارهای کلیدی عملکردش با افزایش اندازه سیستم، تغییر نمی‌یابد.

با توجه به کار آینده، ما طرح‌ریزی می‌کنیم برای (1) تعیین نرخ همگرایی *GRMP-Q* و وابستگی آن به تقاضاهای پردازنده و حافظه، (2) توسعه یک نسخه از پروتکل برای یک محیط ابر ناهمگن که در آن ظرفیت پردازنده و حافظه در سراسر دستگاه‌ها متفاوت است، (3) توسعه یک مکانیسم توزیع‌شده که به‌صورت کارآمد سایت‌های جدید را قرار می‌دهد، (4) ایجاد پروتکل قوی برای خرابی دستگاه‌ها، (5) توسعه نسخه‌های *GRMP* اهداف بیشتری را پشتیبانی می‌کند (مانند تمایز خدمات) و محدودیت (مانند اشتراک مکانی و ضد اشتراک مکانی[۴۴])، (6) توسعه یک اجرای مقیاس‌پذیر از خدمات مخزن دستگاه که انرژی مصرفی را برای خنک‌کننده در نظر بگیرد.

REFERENCES

[1] U.S. EPA, “Report to congress on server and data center energy efficiency public law 109-431,” 2007.
[2] The Climate Group, “Smart 2020: Enabling thelow carbon economy in theinformation age,” June 2008.
[3] Open Data Center Alliance, “Open data center alliance usage: carbon footprint values,” June 2011.
[4] VMWare, “VmwareR distributed power management white paper,” http://www.vmware.com/files/pdf/DPM.pdf.
[5] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, “Server workload analysis for power minimization using consolidation,” in USENIX’09. Berkeley, CA, USA: USENIX Association, 2009, pp. 28–28.
[6] U.S. EPA, “Working group notes from the EPA technical workshop on energy efficient servers and datacenters,” 2007.

[44] Colocation and anti-colocation

[7] F. Wuhib, R. Stadler, and M. Spreitzer, “Gossip-based resource management for cloud environments,” in CNSM 2010, October, pp. 1 –8.
[8] OpenNebula Project Leads, “http://opennebula.org/.”
[9] OpenStack, “http://openstack.org/.”
[10] UC Santa Barbara, “http://appscale.cs.ucsb.edu/.”
[11] VMWare, “http://www.cloudfoundry.com/.”
[12] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, “Dynamic estimation of CPU demand of web traffic,” in ValueTools 2006. New York, NY, USA: ACM, p. 26.
[13] Z. Gong, X. Gu, and J. Wilkes, “PRESS: PRedictive Elastic ReSource Scaling for cloud systems,” in CNSM 2010, October, pp. 9 –16.
[14] D. Meisner, B. T. Gold, and T. F. Wenisch, “PowerNap: eliminating server idle power,” SIGPLAN Not., vol. 44, pp. 205–216, March 2009.
[15] Hewlett-Packard, Intel, Microsoft, Phoenix Technologies Ltd., Toshiba Corporations, “Advanced configuration and power interface specification,” 2010.
[16] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, “Utility-based placement of dynamic web applications with fairness goals,” in IEEE NOMS, April 2008, pp. 9 –16.
[17] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, “A scalable application placement controller for enterprise data centers,” in WWW2007. New York, NY, USA: ACM, 2007, pp. 331–340.
[18] R. Yanggratoke, F. Wuhib, and R. Stadler, “Gossip-based resource allocation for green computing in large clouds (long version),” KTH Royal Institute of Technology, https://eeweb01.ee.kth.se/upload/publications/reports/2011/ TRITA-EE 2011 036.pdf, Tech. Rep. TRITA-EE 2011:036, April 2011.
[19] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and Zipf-like distributions: evidence and implications,” in INFOCOM, vol. 1, 1999, pp. 126 –134.
[20] V. Petrucci, O. Loques, and D. Moss´e, “Dynamic optimization of power and performance for virtualized server clusters,” in ACM SAC 2010, 2010, pp. 263–264.
[21] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, “Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures,” in ICDCS2010, 2010, pp. 62 –73.
[22] B. Speitkamp and M. Bichler, “A mathematical programming approach for server consolidation problems in virtualized data centers,” IEEE TSC, vol. 3, no. 4, pp. 266 –278, 2010.
[23] N. Tolia, Z. Wang, P. Ranganathan, C. Bash, M. Marwah, and X. Zhu, “Unified thermal and power management in server enclosures,” ASME Conference Proceedings, vol. 2009, no. 43604, pp. 721–730, 2009.
[24] M. Cardosa, M. Korupolu, and A. Singh, “Shares and utilities based power consolidation in virtualized server environments,” in IM 2009, pp. 327 –334.
[25] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, “An integrated approach to resource pool management: Policies, efficiency and quality metrics,” in Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on, june 2008, pp. 326 –335.
[26] C. Subramanian, A. Vasan, and A. Sivasubramaniam, “Reducing data center power with server consolidation: Approximation and evaluation,” in HiPC 2010, 2010, pp. 1 –10.
[27] J. Choi, S. Govindan, J. Jeong, B. Urgaonkar, and A. Sivasubramaniam, “Power consumption prediction and power-aware packing in consolidated environments,” Computers, IEEE Transactions on, vol. 59, no. 12, pp. 1640 –1654, 2010.