وتر: پروتکل مراجعه نظیر به نظیر مقیاس پذیر برای برنامه های کاربردی اینترنت

چکیده

مشکل اساسی است که برنامه های کاربردی نظیر به نظیر با آن مواجه است، محل کارآمد گره است که یک آیتم داده مورد نظر را ذخیره می کند. در این مقاله ارائه وتر ارائه شده است، پروتکل مراجعه توزیع شده که به این مشکل می پردازد. وتر تنها حمایت از یک عملیات را فراهم می کند: با توجه به یک کلید، کلید را بر روی یک گره می نگارد. با ارتباط یک کلید با هر آیتم داده، و ذخیره جفت کلید / داده در این گره که کلید برای آن نگاشته شده است به راحتی می توان مکان داده ها را در بالای وتر پیاده سازی نمود. زمانی که گره ها متصل می شوند و سیستم را ترک می کنند، وتر به طور کارآمدی تطبیق می یابد و حتی اگر این سیستم به طور مداوم در حال تغییر باشد می تواند به پرس و جوها پاسخ دهد. نتایج حاصل از تجزیه و تحلیل نظری و شبیه سازی نشان می دهد که وتر مقیاس پذیر است: هزینه های ارتباطات و حالت حفظ شده توسط هر گره به طور لگاریتمی با تعداد گره وتر. مقیاس بندی می شود.

I. مقدمه

سیستم ها و برنامه های کاربردی نظیر به نظیر، سیستم های توزیع شده بدون هیچ گونه کنترل مرکزی و یا سازمان سلسله مراتبی هستند که در آن که هر گره کاربرد را با قابلیت های معادل اجرا می کند. بررسی ویژگی های برنامه های کاربردی نظیر به نظیر نشان دهنده یک فهرست طولانی است: ذخیره سازی افزونه، تداوم، انتخاب سرورهای اطراف، گمنامی، جستجو، ورود و خروج، و نامگذاری سلسله مراتبی. با وجود این مجموعه غنی

از ویژگی ها، عملیات هسته ای در اکثر سیستم های نظیر به نظیر، محل موثر اقلام داده ها است. سهم این مقاله یک پروتکل مقیاس پذیر برای مراجعه در سیستم پویای نظیر به نظیر با ورودها و حرکت های مکرر گره است.

پروتکل وتر فقط از یک عملیات پشتیبانی می کند: با توجه به یک کلید، این کلید بر روی یک گره نگاشته می شود. بسته به نوع کاربرد استفاده کننده از وتر، آن گره ممکن است برای ذخیره یک مقدار در ارتباط با کلید مسئول باشد. وتر از هش های سازگار [12برای انتصاب کلیدها به گره های وتر استفاده می کند. هش برای تعادل بار است، از اینرو هر گره تقریبا تعداد یکسانی از کلیدها را دریافت می کند، و نیاز به حرکت نسبتا کمی از کلید ها دارد زمانی که گره ها متصل می شوند و سیستم را ترک می کنند.

کار قبلی در مورد هش سازگار فرض می کند که هر گره از بسیاری از گره های دیگر در این سیستم آگاه است، یک روش که به خوبی با تعداد زیادی از گره ها مقیاس بندی نمی شود. در مقابل، هر گره وتر نیاز به اطلاعات "مسیریابی" مربوط به تنها چند گره دیگر دارد. از آنجا که جدول مسیریابی توزیع شده است، یک گره وتر به منظور انجام مراجعه با گره های دیگر ارتباط برقرار می کند. در حالت پایدار، در یک سیستم N گره، هر گره تنها اطلاعات مربوط به O (logN) گره دیگر را حفظ می کند، و تمام این مراجعات را توسط O (logN) پیام به گره های دیگر حل می کند. وتر اطلاعات مسیریابی خود را زمانی حفظ می کند که گره ها متصل می شوند و از سیستم خروج می کنند.

یک گره وتر نیاز به اطلاعات مربوط به O (logN) گره دیگر برای مسیریابی کارآمد دارد، اما عملکرد آن وقتی که آن اطلاعات قدیمی باشد، به آرامی تنزل می کند. این مورد در عمل مهم است چرا که که گره ها دلخواه متصل می شوند و ترک می کنند، و حفظ سازگاری O (logN) حالت ممکن است سخت باشد. فقط یک قطعه از اطلاعات در هر گره برای وتر به منظور تضمین مسیریابی درست (هر چند احتمالا آهسته) پرس و جوها نیاز به تصحیح دارد؛ وتر دارای یک الگوریتم ساده برای حفظ این اطلاعات در یک محیط پویا است.

سهم این مقاله، الگوریتم وتر، اثبات صحت آن، و نشان دادن نتایج شبیه سازی قدرت و صلابت این الگوریتم است. همچنین ما برخی از نتایج اولیه را در این مورد که چگونه پروتکل مسیریابی وتر می تواند برای در نظر گرفتن توپولوژی فیزیکی شبکه گسترش یابد، گزارش می دهیم. خوانندگان علاقه مند به استفاده از وتر و چگونگی رفتار وتر در بستر کوچک اینترنت به Dabek و همکاران ارجاع می شوند. [9]. نتایج گزارش شده توسط Dabek و همکاران مطابق با نتایج شبیه سازی ارائه شده در این مقاله است.

بقیه این مقاله به شرح زیر ساختاربندی شده است. بخش دوم وتر را با کار مرتبط مقایسه می کند. بخش سوم ارائه دهنده مدل سیستم است که پروتکل وتر را تحریک می کند بخش چهارم ارائه دهنده پروتکل وتر است و بسیاری از خواص آن را ثابت می کند. بخش پنجم ارائه دهنده شبیه سازی های حمایت کننده از ادعای ما در مورد عملکرد وتر است. در نهایت، سهم ما در بخش شش خلاصه شده است.

II. کار مرتبط

سه ویژگی وتر را از بسیاری دیگر از پروتکل های مراجعه نظیر به نظیر متمایز می کند، سادگی، صحت قابل اثبات، و عملکرد قابل اثبات آن است.

برای روشن شدن مقایسه با کار مربوطه، ما در این بخش یک کاربرد مبتنی بر وتر را فرض می کنیم که کلیدهایی را بر روی مقادیر می نگارد. این مقدار می تواند یک آدرس، یک سند، یا به طور دلخواه آیتم داده باشد. کاربرد مبتنی وتر هر مقدار در گره ای که بر روی آن کلیدی مقدار را می نگارد، ذخیره سازی و پیدا می کند.

DNS خدمات مراجعه را با نام میزبان به عنوان کلید و آدرس های IP و (دیگر میزبان اطلاعات) به عنوان مقدار فراهم می کند. وتر می تواند خدمات مشابه با هش هر یک از نام های میزبان را به یک کلید ارائه دهد [7.] DNS مبتنی بر وتر نیاز به سرور خاصی ندارد، در حالی که DNS عادی متکی بر مجموعه ای از سرورهای ریشه

خاص است DNS .نیاز به مدیریت راهنمای اطلاعات مسیریابی دارد تا (سوابقNS) که برای کاربران جستجو در سلسله مراتب سرور را میسر می سازد؛ وتر به طور خودکار صحت اطلاعات مسیریابی مشابه را حفظ می کند. DNSتنها زمانی کار می کند که نام های ساختاریافته میزبان منعکس کننده مرزهای مدیریتی باشد؛ وتر هیچ ساختار نامگذاری را تحمیل نمی کند. DNS متخصص وظیفه پیدا کردن نام میزبان یا خدمات نامگذاری شده است، در حالی که وتر برای پیدا کردن اشیاء داده هایی می تواند مورد استفاده قرار گیرد که به ماشین های خاص گره خورده اند.

سیستم ذخیره سازی نظیر به نظیر Freenet[5]، [6]، مانند وتر، غیر متمرکز و متقارن است و به طور خودکار زمانی سازگار می شود که میزبان ها ترک و به هم می پیوندند. Freenet مسئولیت اسناد را به سرور های خاص منصوب نمی کند، در عوض، مراجعه شکل جستجو را برای نسخه های کش می گیرد. این کار به Freenetاجازه می دهد تا درجه ای از گمنامی را ارائه دهد، اما مانع از آن می شود که بازیابی مدارک موجود تضمین شود و یا از فراهم مرزهای پایین در بازیابی هزینه ها ممانعت می کند. وتر نامش را فاش نمی کند، اما عمل مراجعه در زمان قابل پیش بینی اجرا می شود و نتیجه همیشه موفقیت و یا ناموفق قطعی است.

سیستم Ohaha از اسناد سازگار نگاشت الگوریتم هش مانند به گره ها، و مسیریابی پرس و جو به سبک Freenet[20.[استفاده می کند. به عنوان یک نتیجه، برخی از نقاط ضعف Freenetرا به اشتراک می گذارد. آرشیو درون حافظه از درخت محاسبه شده خارج از خط برای نگاشت آدرس های منطقی به ماشین هایی که داده ها را ذخیره می کنند استفاده می کند [4].

سیستم Globe [2] دارای خدمات محلی منطقه گسترده برای نگاشت شناسه های شی به مکان اشیاء در حال حرکت است. Globe اینترنت را به عنوان یک سلسله مراتب جغرافیایی، توپولوژیک، و یا حوزه های اداری، به طور موثر با ساختاربندی درخت جستجوی ایستا در سراسر جهان، بسیار شبیه به DNS مرتب می کند. اطلاعات در مورد جسم در یک دامنه خاص برگ ذخیره می شود، و اشاره گر میانبرهای جستجو را ارائه می دهد [25].

سیستم Globe بار زیادی را بر روی ریشه منطقی توسط پارتیشن بندی اشیاء در میان چندین سرورهای ریشه فیزیکی با استفاده از روش هایی مانند هش هدایت می کند.. وتر این تابع هش را به اندازه کافی انجام می دهد تا بتواند به مقیاس پذیری بدون ارتباط با هر سلسله مراتبی دستیابی داشته باشد، هر چند وتر از موقعیت شبکه و .Globe بهره برداری نمی کند.

پروتکل مکان داده های توزیع شده توسعه یافته توسط Plaxton و همکاران [21] شاید نزدیک ترین الگوریتم به پروتکل وتر است. پروتکل مراجعه Tapestry [26]، مورد استفاده در [13] Oceanstore ، یک نوع از الگوریتم .Plaxton است.مانند وتر، تضمین می کند که پرس و جوها بیشتر از یک تعداد لگاریتمی از هاپ ها را که کلید های متعادل هستند، نسازند. مزیت اصلی پروتکل Plaxton بر وتر اینست که، منوط به مفروضات در مورد توپولوژی شبکه، تضمین می کند که پرس و جوها هرگز در فاصله شبکه در بیشتر از یک گره که کلید ذخیره شده است، حرکت نمی کنند. وتر، از سوی دیگر، به طور قابل ملاحظه ای کمتر پیچیده است و پیوندهای همزمان گره ها را هدایت می کند. [23] Pastry یک پروتکل مراجعه مبتنی بر پیشوند است که دارای خواص مشابه با وتر است. مانند Tapestry، Pastry توپولوژی شبکه را برای کاهش پوشیدگی مسیریابی در نظر می گیرد. با این حال، Pastry به این هزینه پروتکل پیوستن شفاف تر می رسد که جدول مسیریابی گره جدید را با استفاده از اطلاعات از گره ها در طول مسیر عرضی با پیام اتصال، مقدار دهی می کند.

CAN از یک فضای-D بعدی مختصات دکارتی (برخی از مقادیر ثابت d) برای پیاده سازی یک جدول هش توزیع شده استفاده می کند که کلیدها را بر روی مقادیر می نگارد [22]. هر گره O(d) حالت را حفظ می کند و $O(dN^{1/d})$.. هزینه مراجعه است. بنابراین، در مقابل وتر، حالت حفظ شده توسط یک گره CAN به اندازه شبکه Nبستگی ندارد، اما هزینه مراجعه سریع تر از .logNافزایش می یابد. اگرD = logN زمان های مراجعه CAN و نیازهای ذخیره سازی با وتر تطابق دارد. با این حال، CAN برای تغییر d به صورت تغییر N طراحی نشده است (و در نتیجه(logN) ، بنابراین این تطبیق فقط برای " N "درست مربوط به d ثابت رخ می دهد.

CAN نیاز به پروتکل نگهداری اضافی برای نگاشت به صورت دوره ای فضای شناسه بر روی گره ها دارد. وتر همچنین دارای این مزیت است که صحت آن در برابر روبرویی با اطلاعات مسیریابی تا حدی نادرست قوی است.

ممکن است رویه مسیریابی وتر به عنوان آنالوگ تک بعدی از سیستم موقعیت Grid تصور شود..[15] (GLS) GLSمتکی بر اطلاعات موقعیت جغرافیایی در دنیای واقعی مسیر پرس و جوها است. وتر گره های خود را به فضای مصنوعی تک بعدی می نگارد که در آن مسیریابی توسط یک الگوریتم مشابه با Grid انجام می شود.

Napster [18] و Gnutella [11] یک عملیات مراجعه را برای پیدا کردن داده ها در یک توزیع مجموعه شده از همتایان ارائه نموده اند. آنها بر اساس کلمات کلیدی تهیه شده توسط کاربر جستجو می کنند، در حالی که وتر داده ها را با شناسه منحصر به فرد جستجو می کند. استفاده از جستجوی کلمه کلیدی، مشکلاتی را در هر دو سیستم ارائه می کند. Napster .از شاخص های مرکزی استفاده می کند که نتیجه ان یک نقطه ناموفق است. Gnutella هر پرس و جو را بر روی کل سیستم جاری می کند، بنابراین، ارتباطات و هزینه های پردازش آن در سیستم های بزرگ بالا است.

وتر به عنوان پایه ای برای تعدادی از پروژه های تحقیقاتی بعدی استفاده می شود. سیستم فایل وتر (CFS) فایلها و فرا داده ها در سیستم های نظیر به نظیر، با استفاده از وتر برای قرار دادن بلوک های ذخیره سازی، ذخیره می کند [9]. تکنیک های تجزیه و تحلیل جدید نشان داده اند که الگوریتم های تثبیت وتر (با تغییرات جزئی) عملکرد خوب مراجعه را به رغم ناموفق مداوم و پیوستن گره ها حفظ می کنند [16]. وتر به عنوان ابزاری برای خدمت به DNS [7] و برای حفظ یک پایگاه داده کلید عمومی توزیع شده برای وضوح نام امن ارزیابی شده است [1].

III. مدل سیستم

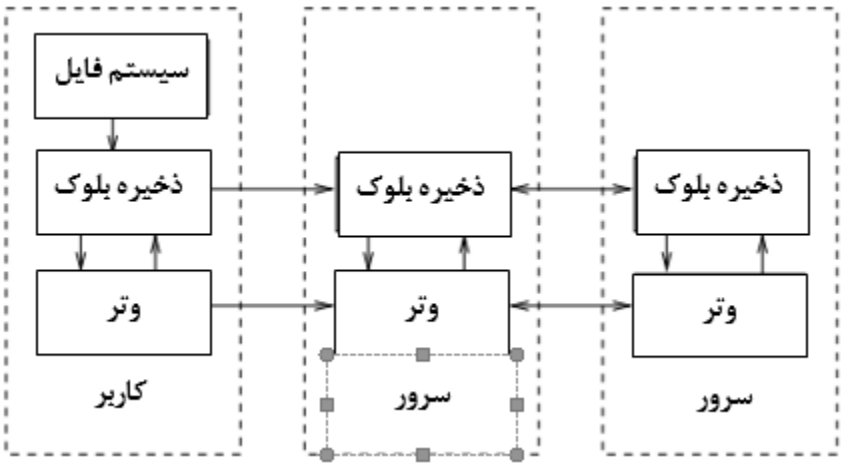
وتر طراحی سیستم های نظیر به نظیر و برنامه های کاربردی را بر اساس آن با پرداختن به این مسائل دشوار ساده می کند:

- تعادل بار: وتر به عنوان یک تابع هش توزیع شده عمل می کند که کلیدهایی را به طور مساوی بر روی گره ها منتشر می کند؛ این کار درجه تعادل بار طبیعی را فراهم می کند.
- عدم تمرکز: وتر به طور کامل توزیع شده است: هیچ گره ای مهم تر از دیگری نیست. این کار نیرومندی را بهبود می بخشد و باعث می شود وتر برای برنامه های کاربردی نظیر به نظیر ضعیف مناسب باشد.
- مقیاس پذیری: هزینه مراجعه وتر زمان ورود تعدادی از گره ها به سیستم رشد می کند، به طوری که حتی سیستم های بسیار بزرگ امکان پذیر می شوند. هیچ تنظیم پارامتری برای رسیدن به این مقیاس بندی مورد نیاز نیست.
- در دسترس بودن: وتر به طور خودکار جدول های داخلی خود را برای منعکس کردن گره های تازه ملحق شده و ناموفق گره ها تنظیم می کند که تضمین می کند که با محرومیت از ناموفق بزرگ در شبکه اساسی، گره مسئول برای یک کلید همیشه می تواند یافت شود. این درست است حتی اگر سیستم در حالت تغییر مداوم باشد.
- نامگذاری انعطاف پذیر: وتر هیچ محدودیتی را در ساختار کلیدهای خود که به دنبال آن است، قائل نمی شود: فضای کلید وتر هموار است. این کار مقدار زیادی از انعطاف پذیری را در چگونگی نگاشت اسامی آنها به کلیدهای وتر ارائه می دهد.

نرم افزار وتر شکلی از کتابخانه را برای پیوند با برنامه های کاربردی که از آن استفاده می کند، اتخاذ می کند. این کاربرد با وتر با دو راه اصلی تعامل دارد. اول، کتابخانه وتر تابع مراجعه (کلید) را فراهم می کند که آدرس IP گره مسئول کلید را ارائه می کند. دوم، نرم افزار وتر در هر گره استفاده از تغییرات را در مجموعه ای از کلیدها

که گره بر عهده دارد، اعلام می کند. این به نرم افزار کاربرد اجازه می دهد تا به عنوان مثال، مقادیر مربوط به خانه های جدید خود را هنگامی که یک گره جدید می پیوندد، انتقال دهد.

این کاربرد استفاده کننده از وتر مسئول ارائه هر گونه احراز هویت مورد نظر، ذخیره، تکثیر، و نامگذاری کاربرپسند داده ها است. فضای کلید هموار وتر پیاده سازی از این ویژگی را آسان می کند. به عنوان مثال، یک برنامه کاربردی می تواند هویت داده ها را با استفاده از ذخیره سازی آن تحت کلید وتر به دست آمده از هش رمزنگاری داده ها تأیید نماید. به طور مشابه، یک برنامه کاربردی می تواند داده ها را با ذخیره سازی آن تحت دو کلید مجزا وتر به دست امده از داده های برنامه در سطح شناسه تکرار نماید.



شکل 1. ساختار نمونه سیستم ذخیره سازی توزیع شده مبتنی بر وتر.

در زیر نمونه هایی از برنامه های کاربردی آمده که برای آن وتر می تواند پایه خوبی را ارائه دهد:

بازتاب همکارانه که در آن ارائه دهندگان متعدد محتوا برای ذخیره و به کارگیری داده های یک دیگر همکاری می کنند. شرکت کنندگان ممکن است، برای مثال، مجموعه ای از پروژه های توسعه نرم افزار باشند که هر یک از آنها باعث انتشار دوره ای می شود. پخش بار کلی به طور مساوی برای همه میزبان های شرکت کننده هزینه های کل سیستم را کاهش می دهد، از اینرو هر یک از شرکت کنندگان فقط نیاز به ارائه ظرفیت برای بار به طور

متوسط دارد نه برای آن بار پیک شرکت کننده. Dabek و همکاران. تحقق این ایده را توصیف نموده اند که از وتر برای نگاشت بلوک های داده بر روی سرورها استفاده می کند؛ این برنامه با تعادل بار وتر، تکرار داده ها، و انتخاب سرور مبتنی بر پوشیدگی تعامل دارد [9].

ذخیره سازی مشترک زمانی برای گره ها با اتصال متناوب. اگر کسی بخواهد اطلاعاتش همیشه در دسترس باشد، اما سرور آن فقط گاه به گاه در دسترس است، می تواند ذخیره داده های دیگران را در حالی که آنها به هم متصل هستند، ارائه دهد، در عوض برای ذخیره داده های خود در جای دیگر زمانی که آنها قطع شده اند. نام داده می تواند به عنوان یک کلید برای شناسایی (زنده) گره وتر مسئول برای ذخیره سازی آیتم داده در هر زمان معین به کار گرفته شود. بسیاری از مسائل مشابه همانند برنامه بازتاب همکارانه بوجود می آیند، هر چند تمرکز در اینجا به جای تعادل بار روی در دسترس بودن است.

شاخص های توزیع شده برای حمایت از جستجوهای کلید واژه مانند Gnutella یا Napster. یک کلید در این کاربرد می تواند از کلمات کلیدی مورد نظر به دست آید، در حالی که مقادیر می توانند لیستی از ماشین آلات ارائه دهنده اسناد با آن کلمات کلیدی باشند.

جستجوی ترکیبی در مقیاس بزرگ مانند ناموفقن کد. در این حالت کلید ها، راه حل های نماینده برای مشکل هستند (مانند کلید های رمزنگاری)؛ وتر این کلیدها را به دستگاه های مسئول آزمایش آنها به عنوان راه حل می نگارد.

چند برنامه کاربردی نظیر به نظیر استفاده کننده از وتر ساخته شده اند. ساختار یک برنامه معمولی در شکل 1 نشان داده شده است. بالاترین لایه، توابع خاص برنامه از قبیل فراداده های سیستم فایل را پیاده سازی می کند. لایه بعدی جدول هش توزیع شده ای را پیاده سازی می کند که برنامه های کاربردی چندگانه برای درج و بازیابی بلوک های داده های مشخص شده با کلید های منحصر به فرد استفاده می کنند. جدول هش توزیع شده

مراقب ذخیره سازی، ذخیره و تکثیر بلوکها است. جدول هش توزیع شده از وتر برای شناسایی گره مسئول ذخیره سازی یک بلوک استفاده می کند، و پس از آن با سرور ذخیره سازی بلوک بر روی آن گره برای خواندن و یا نوشتن بلوک ارتباط برقرار می کند.

IV. پروتکل وتر

در این بخش پروتکل وتر توصیف شده است. پروتکل وتر مشخص می کند که چگونه محل های کلیدها پیدا می شود، چگونه گره های تازه به سیستم می پیوندند، و چگونه از ناموفق (و یا برنامه ریزی شده بهبود می یابند خروج) گره های موجود بازیابی صورت می گیرد. در این مقاله فرض می کنیم که ارتباط در شبکه زیربنایی متقارن است (اگر بتواند به B و سپس B بتواند به A مسیریابی کند) و متعدی (اگر A بتواند به B و سپس B بتواند به C مسیریابی کند، بنابراین A می تواند به C مسیریابی کند).

A. مرور کلی

در قلب آن، وتر محاسبات سریع توزیع شده کلید های نگاشت تابع هش را به گره های مسئول آنها فراهم می کند. وتر کلیدهایی را به گره با درهم گردن سازگار اختصاص می دهد [12] [14]، که دارای بسیاری از ویژگیهای مطلوب است. با احتمال بالا، تابع هش بار را متوازن می کند (تمام گره ها تقریبا تعدادی مشابه از کلیدها را دریافت می کنند). همچنین با احتمال بالا، هنگامی که گره n ام (یا برگ ها) به شبکه می پیوندند، تنها کسر O(1/N) از کلیدها به یک مکان متفاوت منتقل می شود- که به وضوح حداقل لازم برای حفظ یک بار متعادل است.

وتر مقیاس پذیری هش سازگار را با اجتناب از این نیاز را بهبود می بخشد که هر گره، گره های دیگر را می شناسد. یک گره وتر فقط نیاز به مقدار کمی از اطلاعات "مسیریابی" مربوط به گره های دیگر دارد. از آنجا که این اطلاعات توزیع شده است، یک گره تابع هش را با ارتباط با گره های دیگر حل می کند. در شبکه های N-

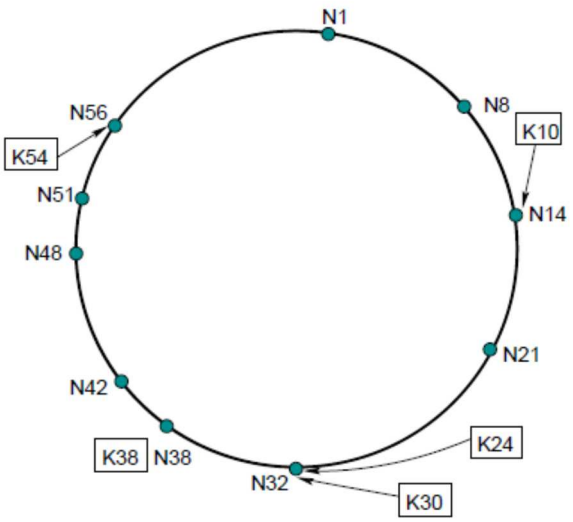
گره، هر گره تنها اطلاعات در مورد O (logN) گره دیگر را را حفظ می کند، و مراجعه نیاز به O (logN) پیام دارد.

B. هش سازگار

تابع هش سازگار به هر گره و کلید یک شناسه کمی m بیتی با استفاده از SHA-1 [10] به عنوان تابع هش پایه اختصاص می دهد. یک شناسه گره برای هش آدرس IP گره انتخاب می شود، در حالی که شناسه کلیدی برای با هش کلید تولید می شود. ما از اصطلاح "کلید" برای اشاره به هر دو کلید اصلی و تصویر آن تحت تابع هش استفاده می کنیم زمانی که این معنا از زمینه روشن خواهد بود. به همین ترتیب، اصطلاح "گره" به گره و شناسه آن تحت تابع هش اشاره می کند. شناسه با طول m باید به اندازه کافی برای ایجاد احتمال دو گره یا هش کلیدها برای همان شناسه ناچیز بزرگ باشد.

هش سازگار کلیدهایی را به گره ها به شرح زیر اختصاص می دهد. شناسه ها روی دایره M2 پیمانه مرتب می شوند. کلید K به اولین گره که شناسه آن برابر با و یا (شناسه) K در فضای شناسه را پیروی می کند، اختصاص می دهد. این گره گره جانشین k نامیده می شود، که توسط جانشین (K) نشان داده می شود. اگر شناسه ها به عنوان یک دایره ای از اعداد از 0 تا $2^m - 1$ نشان داده شوند، بنابراین جانشین (K) اولین گره در جهت عقربه های ساعت از K است. در باقی این مقاله، ما نیز به دایره شناسه به عنوان حلقه وتر اشاره می کنیم.

شکل 2 حلقه وتر با m=6 را نشان می دهد. حلقه وتر دارای 10 گره است و 5 کلید را ذخیره می کند. جانشین شناسه 10 گره 14 است، بنابراین کلید 10 در گره 14 واقع می شود. به طور مشابه، کلیدهای 24 و 30 در گره 32، کلید 38 در گره 38 ، و کلید 54 در گره 56 واقع می شود.



شکل 2. یک دایره شناسه (حلقه) متشکل از 10 گره، ذخیره سازی 5 کلید.

هش سازگار برای مجاز نمودن گره ها برای ورود و ترک شبکه با حداقل اخلال طراحی شده است. برای حفظ نگاشت هش سازگار هنگامی که یک گره n به شبکه می پیوندد، کلید های خاص قبلا منصوب شده به جانشین n در حال حاضر به n اختصاص داده می شود. هنگامی که گره n شبکه را ترک می کند، همه کلید های اختصاص داده شده آن دوباره به جانشین n اختصاص داده می شوند. هیچ تغییرات دیگری در واگذاری کلیدها به گره نباید رخ دهد. در مثال بالا، اگر یک گره به 26 شناسه متصل شود، کلیدی را با شناسه 24 از گره با شناسه 32 ضبط می کند.

نتایج زیر در مقالاتی که هش سازگار را معرفی نموده اند، به اثبات رسیده اند [12] [14]:

قضیه IV.1: برای مجموعه ای از N گره و K کلید، با احتمال بالا داریم:

1. هر گره حداکثر مسئول $(1+\varepsilon)$ K / N کلید است

2. هنگامی که گره (N + 1)ام به شبکه می پیوندد و یا آن را ترک می کند، مسئولیت O (K / N) کلید تغییر می کند (و تنها برای و از گره متصل یا ترک کننده)

هنگامی که هش سازگار همانطور که در بالا توضیح داده شد اجرا می شود، قضیه مرز $\varepsilon = O(\log N)$ را ثابت می کند. مقاله هش سازگار نشان می دهد که ε می تواند به طور دلخواه با داشتن هر یک از اجراهای گره ثابت $\Omega(\log N)$ گره مجازی، هر کدام با شناسه خود به یک ثابت کوچک کاهش یابد. در باقی این مقاله، ما تمامی قیود برحسب کار در هر گره مجازی تجزیه و تحلیل می کنیم. بنابراین، اگر هر گره واقعی v گره مجازی را راه اندازی کند، تمامی قیود باید در v ضرب شود.

عبارت "با احتمال بالا" مقداری بحث را نیاز دارد. تفسیر ساده این است که گره ها و کلید ها به صورت تصادفی انتخاب شده اند، که در مدل غیر خصمانه جهان قابل قبول است. بنابراین توزیع احتمال روی گزینه های تصادفی کلید ها و گره ها است و می گوید که بعید است چنین انتخاب تصادفی یک توزیع نامتعادل را تولید کند. یک مدل مشابه برای تجزیه و تحلیل هش استاندارد اعمال می شود. توابع هش استاندارد داده ها را زمانی که مجموعه ای از کلید های هش شده تصادفی هستند، توزیع می کند. هنگامی که کلیدها تصادفی نیستند، چنین نتیجه ای نمی تواند تضمین شود، در واقع، برای هر تابع هش، مجموعه ای از کلیدها وجود دارد که به طور وحشتناکی توسط تابع هش توزیع شده است (به عنوان مثال، مجموعه ای از کلیدها که همه به یک سطل هش نگاشته می شوند). در عمل، بعید است چنین مجموعه بد بالقوه ای بوجود می آیند. تکنیک هایی [3] برای معرفی اتفاقی بودن در تابع هش توسعه یافته اند؛ با توجه به هر مجموعه ای از کلیدها، ما می توانیم یک تابع هش را به طور تصادفی انتخاب کنیم به طوری که کلیدها به خوبی با احتمال بالا روی انتخاب تابع هش توزیع می شوند. یک تکنیک مشابه می تواند برای هش کردن سازگار اعمال شود؛ بنابراین احتمال بالا در قضیه بالا ادعا شده است. به جای انتخاب یک تابع هش تصادفی ما از الگوریتم SHA-1 استفاده می کنیم که انتظار می رود که خواص خوب توزیعی داشته باشد.

البته، پس از آن که تابع هش تصادفی انتخاب شود؛ یک رقیب می تواند انتخاب یک مجموعه بد توزیع شده از کلید ها را برای توزیع آن تابع هش انتخاب کند. در برنامه ما، یک رقیب می تواند مجموعه بزرگی از کلید ها را

تولید کند و در حلقه وتر تنها آن کلیدهایی را قرار دهد که به یک گره خاص نگاشته می شوند و در نتیجه باعث ایجاد مجموعه ای بد توزیع شده از کلیدها می شود. با این حال، همانند هش کردن استاندارد، ما انتظار داریم که مجموعه ای غیرخصمانه از کلیدها را می توان تجزیه و تحلیل نمود اگر آنها تصادفی باشند. با استفاده از این فرض ما بسیاری از نتایج زیر را به عنوان نتایج "احتمال بالا" اعلام می داریم.

C. موقعیت ساده کلید

در این بخش یک الگوریتم مراجعه وتر ساده اما آهسته را توصیف می کنیم. بخش های بعدی توصیف می کند که چگونه گسترش الگوریتم اساسی برای افزایش بهره وری، و چگونگی حفظ صحت اطلاعات مسیریابی وتر صورت می گیرد.

این مراجعه ها می توانند در حلقه وتر با حالت در هر گره پیاده سازی شوند. هر گره تنها نیاز به دانستن این مورد دارد که چگونه تماس گره جانشین فعلی با دایره شناسه صورت می گیرد. جستجوها برای یک شناسه معین می تواند دور دایره از طریق این اشاره گرهای جانشین بگذرد تا زمانی که یک جفت از گره های که گشاد گشاد قرار گرفته اند با آنها روبرو می شوند؛ مورد دوم در جفت گره ای استکه پرس و جو با آن داده می شود.

شکل 3(a) یک شبه کد را نشان می دهد که مراجعه ساده کلید را پیاده سازی می کند. تماس های از راه دور و منابع مراجعات توسط شناسه گره از راه دور مقدم می شوند، در حالی که مراجع مراجعات محلی و تماس های رویه گره محلی را حذف می کند. بنابراین ()n.foo نشان دهنده تماس رویه از راه دور برای روال foo در گره n است، در حالی که n.bar، بدون پرانتز، یک RPC برای واکشی مراجعات bar از گره n است. نماد (a,b] نشان دهنده بخشی از وتر است که توسط حرکت در جهت عقربه های ساعت از a به دست آمده (اما نه شامل) تا رسیدن به b (و شامل).

شکل 3(b) به عنوان مثالی را نشان می دهد که در آن گره 8 مراجعه برای کلید54 را انجام می دهد. گره 8 find-succesor را برای کلید 54 فراخوانی می نماید که در نهایت جانشین آن کلید، گره 56 را بر می گرداند. پرس و جو هر گره را در دایره ای بین گره های 8 و 56 ویزیت می کند. این نتیجه به همراه معکوس مسیر دنبال شده توسط این پرس و جو برگردانده می شود.

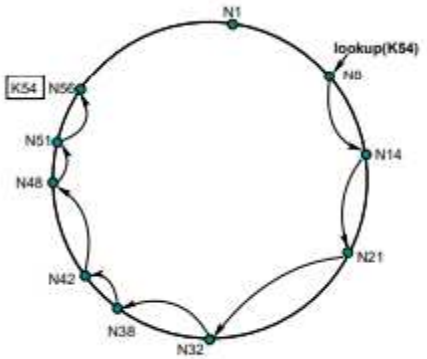
D. موقعیت کلیدی مقیاس پذیر

طرح مراجعه ارائه شده در بخش قبلی از تعدادی از پیام های خطی در تعدادی از گره ها استفاده می کند. برای سرعت بخشیدن به این مراجعات، وتر اطلاعات اضافی مسیریابی را حفظ می کند. این اطلاعات اضافی برای صحت ضروری نیست، که تا زمانی که هر گره جانشین درست خود را بشناسد، به دست می آید.

مانند قبل، در نظر بگیرید m تعداد بیت در شناسه های کلید / گره باشد. هر گره n جدول مسیریابی را تا حدود m ورودی حفظ می کند (خواهیم دید که در واقع تنها O(log n) مجزا هستند)، به نام جدول انگشت. ورود iام در جدول در گره n شامل هویت گره اول می شود که حداقل با 2^{i-1} در دایره شناسه جایگزین می شود، به عنوان مثال، $s = successor(n+2^{i-1})$ که در آن $1 \leq i \leq m$ (و همه حساب ها پیمانه M2 است). ما گره s را انگشت i ام گره n می نامیم و توسط n نشاندهنده ان است. finger[i] (نگاه کنید به جدول I). یک ورودی جدول انگشت شامل شناسه وتر و آدرس IP (و شماره پورت) گره های مربوطه می شود. توجه داشته باشید که انگشت اول n، جانشین بلافاصله n در دایره است؛ برای راحتی ما اغلب به انگشت به عنوان جانشین مراجعه می کنیم.

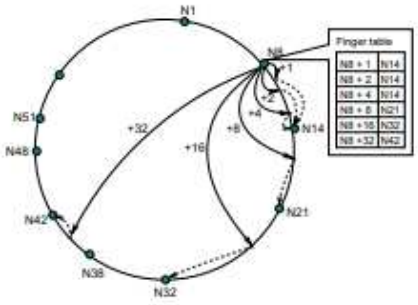
```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    // forward the query around the circle
    return successor.find_successor(id);
```

(a)

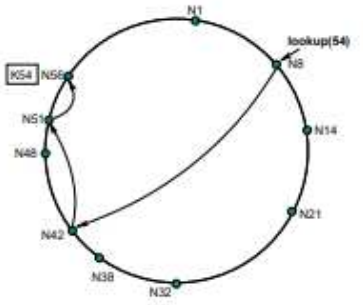


(b)

شکل 3. (a) ساده (اما با سرعت کم) شبه کد برای پیدا کردن گره جانشین id. تماس های رویه از راه دور و مراجعاات مراجعات توسط گره از راه دور مقدم می شوند. (b) این مسیر توسط یک پرس و جو از گره 8 برای کلید 54، با استفاده از شبه کد در شکل 3(a) صورت گرفته است.



(a)



(b)

شکل 4. (الف) مداخل جدول انگشتان گره 8. (ب) روش پرس و جو برای کلید 54 که در گره 8، با استفاده از الگوریتم در شکل 5 شروع می شود.

دستور	تعریف
$finger[k]$	اولین گره روی دایره که جانشین $(n + 2^{k-1}) \bmod 2^m, 1 \leq k \leq m$
جانشین	گره بعدی روی دایره شناسه: finger[a] گره
اولویت دهنده	گره قبلی روی دایره شناسه

جدول I: تعریف مراجعاتهای برای گره n، با استفاده از شناسه های m بیتی.

مثال شکل 4(a) جدول انگشت گره 8 را نشان می دهد. انگشت اول دارای 8 نقطه امتیاز به گره 14 است همانطور که گره 14 گره اول است که $(8+2^0) \bmod 2^6 = 9$.. به طور مشابه، آخرین گره 8 به گره 42 گره 42 اشاره دارد همانطور که گره 42 اولین گره است که $(8+2^5) \bmod 2^6 = 40$..

این طرح دارای دو ویژگی مهم است. اول هر گره تنها اطلاعات تعداد کمی از دیگر گره ها را ذخیره می کند و با پیروی از آن در مورد دایره شناسه بیشتر در مورد گره ها می داند نسبت به مورد گره های دورتر. دوم، یک جدول انگشت گره به طور کلی اطلاعات کافی برای تعیین مستقیم جانشین K کلید دلخواه را شامل نمی شود. به عنوان مثال، گره 8 در شکل 4 (a) نمی تواند جانشین کلید 34 را خود تعیین کند، زمانی که این جانشین (گره 38) در جدول انگشت گره 8 ظاهر شود.

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```

شکل 5. مراجعه کلیدی مقیاس پذیر با استفاده از جدول انگشت.

شکل 5 شبه کد عملیات find_succesor، توسعه یافته برای استفاده از جداول انگشت را نشان می دهد. اگر id بین n و جانشین آن بیافتد، find_succesor تمام می شود و گره n جانشین خود را برمی گرداند. در غیر این صورت، n جدول انگشتان خود را برای گره n’ جستجو می کند که بلافاصله به id اولویت می دهد و

find_succesor را در n’ فراخوانی می کند. دلیل این انتخاب n’ اینست که n’ به id نزدیک تر است، در ادامه در مورد شناسه دایره در منطقه id بیشتر خواهیم دانست.

به عنوان مثال، دایره وتر در شکل 4 (b) را در نظر بگیرید، و فرض کنید گره 8 می خواهد جانشین 54 کلید را پیدا کند. از آنجا که بزرگترین انگشت گره 8 که قبل از 54 می باشد، گره 42 است، گره 8 از گره 42 برای حل و فصل پرس و جو درخواست می کند. در عوض، گره 42 بزرگترین انگشت در جدول انگشتان خود را که قبل از 54 است را تعیین خواهد کرد، به عنوان مثال، گره 51. در نهایت، گره 51 کشف جانشین خود خواهد کرد، گره 56، 54 کلید، موفق، و به این ترتیب گره 56 را به گره 8 بازمی گرداند.

از آنجا که هر گره دارای ورودی های انگشت به توان دو فواصل در اطراف دایره شناسه است، هر گره می تواند پرس و جو را حداقل در نیمه راه باقی مانده در امتداد فاصله بین گره و شناسه هدف ارسال نماید. از این شهود قضیه زیر را خواهیم داشت:

قضیه IV.2: با احتمال بالا، تعداد گره که باید به تماس برای پیدا کردن یک جانشین در شبکه های N-گره O (logN)) است.

اثبات: فرض کنید که گره n تمایل دارد یک پرس و جو را برای جانشین k حل و فصل نماید. فرض کنیم P گره ای باشد که بلافاصله قبل از k قرار دارد. ما تعدادی از مراحل پرس و جو را برای رسیدن به p تجزیه و تحلیل می نماییم.

به یاد بیاورید که اگر n ≠ P، آنگاه n پرس و جوی خود را به نزدیکترین اولویت دهنده K در جدول انگشت خود ارسال می کند. در نظر بگیرید که i را به گونه ای که گره p در بازه $[n+2^{i-1}, n+2^i)$. از آنجایی که این فاصله خالی است (شامل p می شود)، گره n با انگشت i ام، اولین گره f در این فاصله تماس دارد. مسافت (تعداد شناسه ها) بین n و f حداقل 2^{i-1} است. اما p و f در هر دو بازه $[n+2^{i-1}, n+2^i)$ که به

معنی فاصله بین آنها، اکثراً 2^{i-1} است. این f به p نزدیک تر است تا به n، یا معادل، که مسافت از f تا p در بیشتر از نیمی از فاصله از n تا p است.

اگر فاصله میان هدایت گره پرس و جو و پیش نیاز P در هر گام نیم است، و اکثراً 2^m در ابتدا است، بعد از آن در m مرحله فاصله یک خواهد بود، به این معنی که ما به نقطه P رسیده ایم.

در واقع، همانطور که در بالا بحث شد، ما فرض می کنیم که شناسه های گره و کلید تصادفی می باشند. در این مورد، تعدادی از ارسال های لازم با احتمال بالا O (logN) خواهد بود. پس از logN 2 ارسال، فاصله بین گره پرس و جو در حال حاضر و کلید k حداکثر $2^m/N^2$. کاهش می یابد. احتمال اینکه هر گره دیگر در این فاصله اکثراً 1/N است، که قابل اغماض است. بنابراین، گام بعدی ارسال گره مورد نظر را پیدا می کند.

در بخش گزارش دهنده نتایج تجربی ما (بخش V)، ما مشاهده خواهیم کرد (و توجیه می کنیم) که زمان مراجعه متوسط log N ½ است.

اگر چه جدول انگشت شامل اتاق برای m ورودی می شود، در واقع تنها O (logN انگشت نیاز به ذخیره شدن دارد. همانطور که ما در استدلال در بالا اثبات نمودیم، هیچ گره ای به احتمال زیاد در فاصله $2^m/N^2$ برای گره های دیگر وجود ندارد. بنابراین، انگشت گره i ام، برای هر m-2 log N ≤ l برابر جانشین بلافاصله گره با احتمال بالا خواهد بود و لازم نیست جداگانه ذخیره شود.

E. عملیات و ناموفق های پویا

در عمل، وتر نیاز به مقابله با گره های پیوسته به سیستم و با گره هایی دارد که ناموفق می خورند و یا داوطلبانه ترک می کنند. این بخش توضیح می دهد که چگونه وتر این شرایط را هدایت می کند.

E.1 پیوند و ثبات گره

به منظور اطمینان حاصل نمودن از اینکه این مراجعات به درستی همانند مجموعه گره های شرکت تغییر می کنند، وتر باید اطمینان حاصل کند که هر اشاره گر جانشین گره به روز است. این کار با استفاده از پروتکل "ثبات" انجام می شود که هر گره دوره ای در پس زمینه اجرا می شود و اینکه وتر جداول انگشت و جانشین اشاره گر را به روز رسانی می کند.

شکل 6 شبه کد برای پیوندها و ثبات را نشان می دهد. هنگامی که اولین گره n شروع می شود، خواستار $n.join(n')$ می شود که در آن n' هر گره وتر شناخته شده است یا n.create() برای ایجاد یک شبکه جدید وتر است. تابع join() از n' می خواهد تا جانشین بلافاصله n پیدا کند. به خودی خود، join() بقیه شبکه آگاه از n را ایجاد نمی کند.

```
// create a new Chord ring.
n.create()
    predecessor = nil;
    successor = n;

// join a Chord ring containing node n'.
n.join(n')
    predecessor = nil;
    successor = n'.find_successor(n);

// called periodically. verifies n's immediate
// successor, and tells the successor about n.
n.stabilize()
    x = successor.predecessor;
    if (x ∈ (n, successor))
        successor = x;
    successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
    if (predecessor is nil or n' ∈ (predecessor, n))
        predecessor = n';

// called periodically. refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
    next = next + 1;
    if (next > m)
        next = 1;
    finger[next] = find_successor(n + 2^(next-1));

// called periodically. checks whether predecessor has failed.
n.check_predecessor()
    if (predecessor has failed)
        predecessor = nil;
```

شکل 6. شبه کد برای پایداری

هر گره stabilize() را به صورت دوره ای به تازگی در مورد گره های پیوسته اجرا می شود. هر گره زمانی n stabilize() را اجرا می کند، جانشین آن را برای پیش نیاز جانشین،p، فراخوانی می کند و تصمیم می گیرد که آیا P باید جانشین n باشد یا خیر. این مورد در صورتی وجود داشت که P گره اخیرا به سیستم پیوسته باشد. علاوه بر این، stabilize() گره جانشین n را برای وجود n اطلاع می دهد که ارائه دهنده جانشین شانس برای تغییر پیش نیاز آن برای n است. جانشین این کار را فقط در صورتی انجام می دهد که بداند از هیچ پیش نیاز نزدیک تر از n اطلاع نداشته باشد.

هر گره به صورت دوره ای fix fingers را فراخوانی می کند تا مطمئن شود که ورودی های جدول انگشت صحیح هستند؛ این همان چگونگی مقداردهی جداول انگشت آن توسط گره های جدید و چگونگی گنجاندن گره های جدید در جداول انگشتان آنها توسط گره های موجود است. هر گره نیز check predecessor را به صورت دوره ای اجرا می کند، تا اشاره گر پیش نیاز گره مشخص شود اگر پیش نیاز n ناموفق باشد، این کار پذیرفتن پیش نیاز جدید در خبر سازی را میسر می سازد.

به عنوان یک مثال ساده، فرض کنید گره n به سیستم می پیوندد و ID آن بین گره های np و ns نهفته است. در فراخوان برای join()، ns n را به عنوان جانشین خود بدست می آورد. گره ns، زمانی که توسط n مطلع می شود، n را به عنوان پیش نیاز خود بدست می آورد. هنگامی که np stabilize() بعدی را اجرا می کند، از ns برای پیش نیاز خود (که در حال حاضر n است) درخواست می کند؛ پس از آن n np را به عنوان جانشین خود بدست می آورد. در نهایت، np n اطلاع رسانی می کند ، و n np را به عنوان پیش نیاز خود بدست می آورد. در این مرحله، تمام اشاره گرهای پیش نیاز و جانشین درست هستند. در هر گام در این فرآیند، ns از np با استفاده از اشاره گرهای جانشین قابل دسترسی است. این بدین معنی است که مراجعات همزمان با پیوستن مختل نمی شوند. شکل 7 روش پیوستن را نشان می دهد هنگامی که ID n 26 است. و شناسه های ns و np 21 و 32، هستند.

به محض این که اشاره گر جانشین درست باشد، فراخوانی ها برای ()find_succesor گره های جدید را منعکس خواهد کرد. گره های تازه ملحق شده که در جداول انگشت گره های دیگر هنوز منعکس نشده اند ممکن است سبب شوند تا ()find_succesor ابتدائاً به هدف بنشیند، با این حال حلقه در الگوریتم مراجعه به دنبال جانشین اشاره گرihd ([1] finger) از طریق گره های تازه ملحق شده است تا زمانی که پیش نیاز صحیح برسد. در نهایت ()fix_succesor ورودی های جدول انگشت را با از بین بردن نیاز برای این اسکن های خطی تنظیم می کند.

نتیجه زیر که در [24] ثابت شده است، نشان می دهد که حالت متناقض ناشی از پیوندهای همزمان، گذرا است.

قضیه IV.3: اگر هر دنباله ای از عملیات های پیوستن با پایداری اجرا شود، آنگاه در زمانی پس از آخرین پیوستن، اشاره گرهای جانشین یک چرخه را در تمام گره ها در شبکه تشکیل می دهند.

به عبارت دیگر، بعد از مدتی هر یک از گره ها با دنبال نمودن اشاره گرهای جانشین قادر به رسیدن به هر گره در شبکه است.

طرح ثبات اضافه کردن گره به حلقه وتر را به روشی تضمین می کند که قابلیت دسترسی گره های موجود، حتی در مواجهه با پیوند همزمان و پیام های از دست رفته و ضبط شده حفظ می کند. این پروتکل ثبات به خودی خود یک سیستم وتر را که به چند چرخه غیرمتصل تقسیم شده است تصحیح نمی کند، یا یک چرخه تک که چند بار اطراف فضای شناسه حلقه می زند. این موارد آسیب شناختی می تواند توسط هر توالی معمولی از پیوندهای گره ها تولید شود. در صورت تولید، این موارد را می توان شناسایی و توسط نمونه برداری تناوبی توپولوژی حلقه ترمیم نمود [24].

E.2 تاثیر پیوندهای گره ها در این مراجعات

در این بخش، ما تاثیر پیوندهای گره ها در این مراجعات را در نظر می گیریم. ما برای اولین بار صحت را در نظر می گیریم. اگر پیوستن گره ها روی برخی از مناطق حلقه وتر تاثیر بگذارد یک مراجعه که قبل از اینکه ثبات رخ دهد، به پایان رسیده، می تواند نشاندهنده یکی از سه رفتار باشد. مورد مشترک اینست که همه ورودی های جدول انگشت مرتبط با مراجعه جریان منطقی هستند، و این مراجعه جانشین درست را در O (logN) مرحله می یابد. مورد دوم که در آن اشاره گرهای جانشین درست هستند، اما انگشت ها نادرست می باشند. این نشاندهنده مراجعات صحیح است، اما ممکن است کندتر باشد. در مورد آخر، گره ها در منطقه متاثر دارای اشاره گرهای جانشین نادرست هستند یا کلیدها در عین حال ممکن است به گره های تازه ملحق شده مهاجرت کنند و مراجعه ممکن است ناموفق شود. نرم افزار لایه بالاتر که از وتر استفاده می کند اعلام می کند که داده های مورد نظر یافت نشده اند، و دارای گزینه درحال تلاش مجدد مراجعه پس از یک مکث است. این مکث می تواند کوتاه مدت باشد، از اینرو این تثبیت اشاره گرهای جانشین را به سرعت رفع می کند.

حال، عملکرد را در نظر می گیریم. هنگامی که ثبات کامل باشد، گره های جدید هیچ تاثیری فراتر از افزایش N در زمان مراجعه O (logN) ندارند اگر تثبیت هنوز تکمیل نشده باشد، ورودی جدول انگشت برای گره های موجود نمی توانند منعکس کننده گره های جدید باشند. توانایی ورودی های انگشت برای حمل پرس و جوها در فواصل بلند در اطراف حلقه شناسه دقیقا بستگی ندارد به اینکه ورودی ها به کدام گره ها اشاره می کنند؛ به استدلال نصف کردن فاصله تنها به فاصله فضای ID بستگی دارد. بنابراین این واقعیت که ورودی های جدول انگشت ممکن است منعکس کننده گره های جدید نباشند، تاثیر قابل توجهی روی سرعت مراجعه ندارد. راه اصلی که در آن گره های به تازگی پیوسته می توانند روی سرعت مراجعه تاثیر داشته باشند اگر IDهای گره های جدید بین پیش نیاز هدف و هدف باشند. در این مورد مراجعه، باید از طریق گره های متداخل، یک بار در زمان فرستاده شود. اما اگر تعداد بسیار زیادی از گره ها به سیستم بپیوندند، تعداد گره های بین دو گره قدیمی به احتمال زیاد بسیار کم است، بنابراین تاثیر در مراجعه به آنها نیز قابل اغماض است. بعبارت دیگر، ما می

توانیم نتیجه زیر را بیان کنیم. ما حلقه وتر را پایدار می نامیم اگر همه جانشین ها آن و اشاره گرهای انگشت آن درست باشند.

قضیه IV.4: اگر ما یک شبکه پایدار را با N گره با اشاره گرهای انگشت درست اتخاذ کنیم، و یکی دیگر از مجموعه های n گره ای به شبکه بپیوندد، و تمام اشاره گرهای جانشین (اما شاید نه همه اشاره گرهای انگشت) درست باشند، آنگاه این مراجعات هنوز هم زمان O (logN) را با احتمال بالا به خود می گیرد.

اثبات: مجموعه اصلی انگشتان، در زمان O (logN)، پرس و جو را به پیش نیاز قدیمی گره درست می آورد. با احتمال بالا، حداکثر O (logN) گره جدید بین هر دو گره قدیمی زمین می شوند. بنابراین تنها O (logN) گره جدید باید همراه با اشاره گرهای جانشین برای گرفتن از پیش نیاز قدیمی به پیش نیاز جدید حرکت کند.

به طور کلی، تا زمانی که تنظیم انگشتان اتخاذ شود که کمتر از زمانی است که شبکه در اندازه دو برابر می شود، مراجعات را به اتخاذ O (logN) گره ادامه خواهند داد. ما می توانیم به چنین تنظیمی بارها و بارها با انجام مراجعات برای به روز رسانی انگشتان خود دستیابی پیدا کنیم. چنین بر می آید که این مراجعات تا زمانی عمل می کنند که $\Omega(\log^2 N)$ رند کردن ثبات بین هر پیوند گره N اتفاق می افتد.

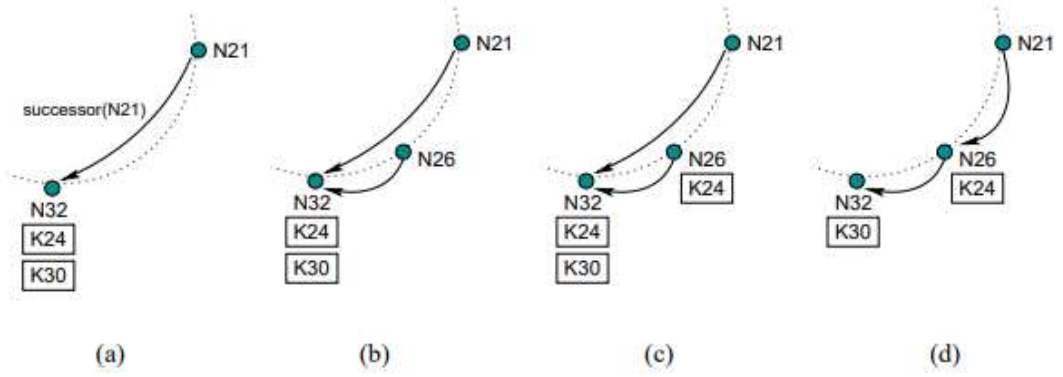
E.3 ناموفق و تکرار

صحت پروتکل وتر، متکی بر این واقعیت است که هر گره جانشین خود را می شناسد. با این حال، این ثابت می تواند به خطر بیافتد اگر گره ناموفق باشد. به عنوان مثال، در شکل 4، اگر گره های 14، 21، و 32 به طور همزمان ناموفق باشند، گره 8 نمی داند که گره 38 در حال حاضر جانشین آن است، چرا که هیچ انگشت اشاره ای به به 38 ندارد. یک جانشین نادرست به مراجعات نادرست منجر می شود. یک پرس و جو برای 30 کلید راه اندازی شده توسط گره 8 را در نظر بگیرید. گره 8 به گره 42 بازخواهد گشت، گره اول آن را از جدول انگشت خود به جای جانشین درست، گره 38 می شناسد.

برای افزایش استحکام، هر گره وتر یک فهرست جانشین از اندازه r را که شامل اولین جانشینان گره r است حفظ می کند. اگر جانشین بلافاصله گره پاسخ ندهد، این گره می تواند جایگزین ورودی دوم در فهرست جانشین آن شود. تمامی جانشینان r بابه طور همزمان به منظور اخلال در حلقه وتر ناموفق می شوند، یک رویداد است که می تواند با مقادیر میانی r بسیار غیر محتمل باشد. با فرض اینکه هر گره به طور مستقل با احتمال p ناموفق است، احتمال اینکه که تمام جانشینان r به طور همزمان ناموفق باشند تنها در این p^r است. افزایش r باعث می شود سیستم قوی تر شود.

هدایت فهرست جانشین نیاز به تغییرات جزئی در شبه کد در شکل های 5 و 6 دارد یک نسخه اصلاح شده از رویه ثبات در شکل 6 فهرست جانشین را نگهداری می کند. لیست های جانشین به شرح زیر تثبیت می شوند: گره n با لیست خود با جانشین خود s توسط کپی نمودن جانشین s، با از بین بردن ورودی گذشته، و وصل شدن به ابتدا، تلفیق می شود. اگر گره n اعلام کند که جانشین آن ناموفق است، با اولین ورودی زنده خود در فهرست جانشین آن جایگزین می شود و لیست جانشین خود را با جانشین جدید خود تلفیق می کند. در آن نقطه، n می تواند مراجعات های معمولی را برای کلیدهایی هدایت کند که برای آن گره ناموفق جانشین گره جدید شده است. همانطور زمان می گذرد، fix_fingers و stabilize ورودی های جدول انگشت و ورودی های فهرست جانشین را با اشاره به گره ناموفق تصحیح می کنند.

یک نسخه اصلاح شده از رویه closest_preceding_node در شکل 5 نه تنها جدول انگشت بلکه لیست جانشین را برای فوری ترین پیش نیاز id جستجو می کند. علاوه بر این، شبه کد باید برای رسیدگی به گره های ناموفق ارتقا یابد. اگر یک گره در طول روش find_succesor ناموفق باشد، مراجعه ، پس از یک زمان استراحت، با امتحان نمودن بهترین پیش نیاز بعدی در میان گره ها در جدول انگشت و لیست جانشین صورت می گیرد.



شکل 7. مثال نشان دادن عملیات پیوستن. گره 26 به سیستم بین گره های 21 و 32 می پیوندد. کمان ها نشان دهنده رابطه جانشین هستند. (الف) حالت اولیه: گره 21 به گره 32 اشاره می کند (ب) گره 26 به جانشین خود اشاره می کند (به عنوان مثال، گره 32) و به آن اشاره می کند (ج) گره 26 تمام کلیدهای کمتر از 26 را از گره 32 کپی می کند (د) رویه تثبیت جانشین گره 21 را به گره 26 به روز رسانی می کند.

نتایج زیر استحکام پروتکل وتر را را تعیین کمیت می نمایند، با نشان دادن اینکه نه موفقیت و نه عملکرد مراجعات وتر به احتمال زیاد حتی توسط شکست های عظیم ناموفق به طور همزمان. هر دو نظریه فرض می کند که طول لیست جانشین برابرست با r = (logN).

قضیه IV.5: اگر ما از یک لیست جانشین با طول r = $\Omega(\log N)$ در شبکه استفاده کنیم که در ابتدا با ثبات است، و سپس هر گره با مواجه با احتمال 1/2 ناموفق شود، آنگاه با احتمال بالا find_succesor نزدیک ترین جانشین زنده خود را به به کلید پرس و جو بر می گرداند.

اثبات: قبل از اینکه هر گره ناموفق باشد، هر گره از جانشینان فوری خود آگاه است. احتمال اینکه تمام این جانشینان ناموفق شوند، $(1/2)^r$ است، بنابراین با احتمال بالا در هر گره از جانشین زنده فوری خود آگاه است. همانطور که در بخش گذشته بحث کردیم، در صورت ثابت بودن که هر گره از جانشین بلافاصله خود آگاه

است، برقرار باشد، آنگاه همه پرس و جوهای به درستی مسیریابی می شوند، از آنجا که هر گره به جز پیش نیاز فوری پرس و جو، حداقل دارای یک گره بهتر است که به آن پرس و جو را ارسال می کند.

قضیه IV.6: در یک شبکه که در ابتدا پایدار است ، اگر هر گره با احتمال 2/1 ناموفق باشد، آنگاه زمان مورد انتظار برای اجرای find_succesor برابر O (logN) است.

اثبات: با توجه به محدودیت های فضایی، ما اثبات این نتیجه را که می تواند در گزارش فنی یافت شود را حذف می کنیم [24].

تحت برخی از شرایط ممکن است قضایای قبلی به گره های ناموفق مخرب و همچنین ناموفق تصادفی اعمال شود. یک رقیب ممکن است قادر باشد تا مجموعه های از گره های ناموفق را ایجاد کند، اما هیچ کنترلی بر انتخاب مجموعه ندارد. به عنوان مثال، رقیب ممکن است تنها قادر به تحت تاثیر قرار دادن گره ها در منطقه خاص جغرافیایی باشد یا تمام گره هایی که از لینک های دسترسی خاص، و یا همه گره هایی استفاده می کنند که دارای پیشوند آدرس IP خاص هستند. همانطور که در بالا مورد بحث قرار گرفت ، به دلیل اینکه IDهای وتر گره توسط توسط هش آدرس IP تولید می شوند، IDهای این گره های ناموفق به طور موثر تصادفی خواهند بود، به همان اندازه مورد ناموفق تجزیه و تحلیل شده در بالا.

مکانیزم لیست جانشین نیز به نرم افزار لایه های بالاتر کمک می کند تا تکرار داده ها را صورت دهد. یک کاربرد معمول استفاده کننده از وتر ممکن است کپی داده های مرتبط با یک کلید را در k گره جانشین کلید ذخیره نماید. این واقعیت که یک گره وتر ردیابی جانشینان r را نگه می دارد، بدان معنی است که می تواند نرم افزار لایه بالا را اطلاع رسانی نماید هنگامی که جانشینان می آیند و می روند و به این ترتیب زمانی که نرم افزار باید اطلاعات را به کپی های جدید منتشر نماید.

E.4 خروج داوطلبانه گره

از آنجا که وتر در مواجهه با خرابی ها مستحکم است، یک گره که به طور داوطلبانه از سیستم خروج می کند می تواند به عنوان یک ناموفق گره ای در نظر گرفته شود. با این حال، دو پیشرفت می تواند عملکرد وتر را بهبود دهد زمانی که گره ها داوطلبانه ترک می کنند. اول، یک گره n که در حال ترک است ممکن است کلیدهای خود را به جانشینان خود را قبل از حرکت انتقال دهد. دوم، n به پیش نیاز خود p و جانشین s قبل از خروج اطلاع می دهد. به نوبه خود، گره p، n را از لیست جانشین خود حذف می کند، و گره آخر را در فهرست جانشین n به لیست خود اضافه می کند. به طور مشابه، گره s پیش نیاز خود را با پیش نیاز n جایگزین می کند. در اینجا ما فرض می کنیم که n پیش نیاز خود را به s، و گره آخر را در لیست جانشین خود به p می فرستد.

F. تجزیه و تحلیل واقعی تر

تحلیل ما برخی از بینش ها را به رفتار سیستم وتر می دهد. اما در عمل ناکافی است. این نظریه های ثابت شده در بالا فرض می کند که حلقه وتر در حالت پایدار شروع می شود و پس از آن اتصال و یا وقوع خرابی تجربه می شود. در عمل، یک حلقه وتر هرگز در حالت پایدار نخواهد بود؛ به جای آن، اتصال و خروج به طور مداوم رخ می دهد، با الگوریتم تثبیت. این حلقه قبل از اینکه تغییرات جدید رخ دهد، هیچ زمانی برای ثبات ندارد. الگوریتم های وتر را می توان در تنظیم کلی تر مورد تجزیه و تحلیل قرار داد. کار دیگر [16] نشان می دهد که در صورتی که پروتکل تثبیت در یک سرعت خاص به اجرا در آید (بستگی به سرعتی دارد که در آن گره می پیوندد و ناموفق می شود) و سپس حلقه وتر و به طور مداوم تقریبا در " حالت پایدار " باقی می ماند که در آن مراجعات سریع و صحیح هستند.

نتایج V. شبیه سازی

در این بخش، ما پروتکل وتر را توسط شبیه سازی ارزیابی می کنیم. شبیه ساز سطح بسته از الگوریتم مراجعه در شکل 5 استفاده می کند که با لیست جانشین توضیح داده شده در بخش IV-E.3 و الگوریتم ثبات در شکل 6 گسترش یافته است.

A. شبیه ساز پروتکل

پروتکل وتر را می توان در سبک تکرار شونده و یا بازگشتی اجرا شود. در سبک تکرار شونده، یک گره در حل و فصل مراجعه تمام ارتباطات را آغاز می کند: مجموعه ای از گره ها را برای کسب اطلاعات از جداول انگشتان خود، هر بار با حرکت نزدیکتر حلقه وتر به جانشین مورد نظر درخواست می کند. در سبک بازگشتی، هر گره میانی، درخواست خود را به گره بعدی تا زمانی ارسال می کند که به جانشین می رسد. این شبیه ساز، پروتکل وتر را در سبک تکرار شونده پیاده سازی می کند.

در طول هر مرحله ثبات، یک گره جانشین بلافاصله خود و یک ورودی دیگر در فهرست جانشین خود و یا جدول انگشت را به روز رسانی می کند. بنابراین، اگر لیست جانشین یک گره و جدول انگشت حاوی مجموع K ورودی منحصر به فرد باشد، هر یک از ورودی ها در هر دور ثبات kتجدید می شود. مگر آن که در غیر این صورت مشخص شده باشد، اندازه لیست جانشین یک است، یعنی یک گره فقط جانشین بلافاصله خود را می شناسد. علاوه بر بهینه سازی شرح داده شده در بخش IV-E.4، این شبیه ساز بهینه سازی دیگری را پیاده سازی می کند. زمانی که پیش نیاز گره n تغییر می کند، n به پیش نیازهای قدیمی خود p در مورد پیش نیاز جدید p′ اطلاع می دهد. این اجازه می دهد تا p جانشین خود را برای p′ بدون انتظار برای دور ثبات آینده تنظیم نماید.

تاخیر هر یک از بسته به طور نمایی توزیع شده با میانگین 50 میلی ثانیه است. اگر یک گره n نتواند با یکی دیگر از گره های n′ در عرض 500 میلی ثانیه تماس برقرار کند، n نتیجه گیری می کند که n′ ترک کرده

است و یا ناموفق است. اگر n′ یک ورود در لیست جانشین n یا جدول انگشت باشد، این ورودی برداشته می شود. در غیر این صورت n به گره ای که از آن در مورد n′ آموخته که n′ رفته است، اطلاع رسانی می کند. هنگامی که یک گره در مسیر مراجعه خراب شود، گره ای که مراجعه را آغاز کرده برای پیشرفت به نزدیکترین انگشت بعدی قبل از کلید هدف تلاش می کند

این مراجعه در صورتی موفق در نظر گرفته می شود که به جانشین کنونی کلید مورد نظر برسد. این کمی خوش بینانه است: در یک سیستم واقعی، ممکن است دوره های زمانی وجود داشته باشد که در ان جانشین واقعی یک کلید هنوز داده های مرتبط با کلید را از جانشین قبلی به دست نیاورده است. با این حال، این روش اجازه می دهد تا بر روی توانایی وتر برای انجام این مراجعات، به جای توانایی کاربرد لایه بالاتر برای حفظ قوام داده های آن تمرکز صورت گیرد.

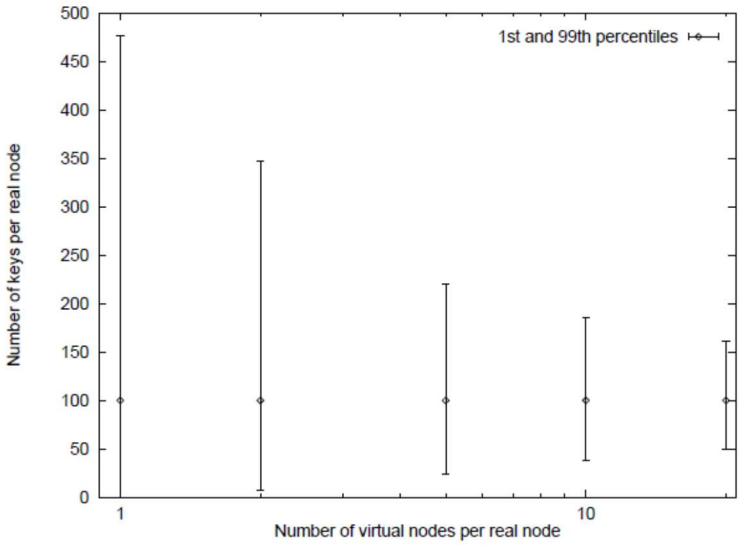
B. تعادل بار

ما برای اولین بار توانایی هش سازگار برای اختصاص دادن کلیدها به گره ها به طور مساوی را در نظر می گیرید. در یک شبکه با N گره و K کلید، ما خواهان محکم بودن توزیع کلیدها به گره های در اطراف N/K هستیم.

ما یک شبکه متشکل از 104 گره را در نظر می گیریم و تعداد کل کلیدها از 105 تا 106 با افزایش 105 متغیر است. برای هر تعداد کلید، ما 20 آزمایش با دانه های تولیدکننده عدد مختلف تصادفی را با شمارش تعداد کلیدهای اختصاص داده شده به هر گره در هر آزمایش اجرا می کنیم. شکل 8 (n) نمودار متوسط و اول و 99 صدک تعداد کلیدها در هر گره است. تعداد کلیدها در هر گره نشاندهنده تغییراتی بزرگ است که به طور خطی با تعداد کلیدها افزایش می یابد. برای مثال، در همه موارد برخی از گره ها هیچ کلیدی را ذخیره نمی کنند. برای روشن شدن این مورد، شکل 8 (b) نمودار تابع چگالی احتمال (PDF) تعداد کلید در هر گره است

زمانی که 5 105 کلید ذخیره شده در شبکه وجود دارد. حداکثر تعداد گره های ذخیره شده توسط هر گره در این مورد، 457، و یا 09:01 است. برای مقایسه، 04:06 مقدار متوسط است.

یکی از دلایل برای این تغییرات این است که شناسه های گره به طور یکنواخت فضای کامل شناسه را پوشش نیمی دهند. از منظر یک گره تک، میزان حلقه ای که دارد توسط فاصله به پیش نیاز فوری آن تعیین می شود. فاصله تا هر یک از n-1 گره دیگر به طور یکنواخت روی محدوده [0، m] توزیع شده است و ما به حداقل این مسافت علاقه مند هستیم.



شکل 9. اولین و 99امین صدک تعداد کلیدها در هر گره به عنوان یک تابع از گره های مجازی نگاشته شده به به یک گره واقعی. این شبکه دارای 104 واقعی گره است و 106 کلید را ذخیره می کند.

این یک واقعیت استاندارد است که توزیع این حداقل، با توزیع نمایی با میانگین $2^m/N$ تقریب زده می شود. بنابراین، برای مثال، منطقه متعلق به دو برابر مقدار میانگین ($2^m/N$) با احتمال e^{-2} تجاوز می کند.

وتر تعدادی از کلید ها در هر گره توسط کلید های مرتبط با گره های مجازی، و نگاشت گره های متعدد مجازی (با شناسه نامربوط) به هر یک از گره های واقعی، یکنواخت تر می سازد. این کار پوشش یکنواخت تری از فضای

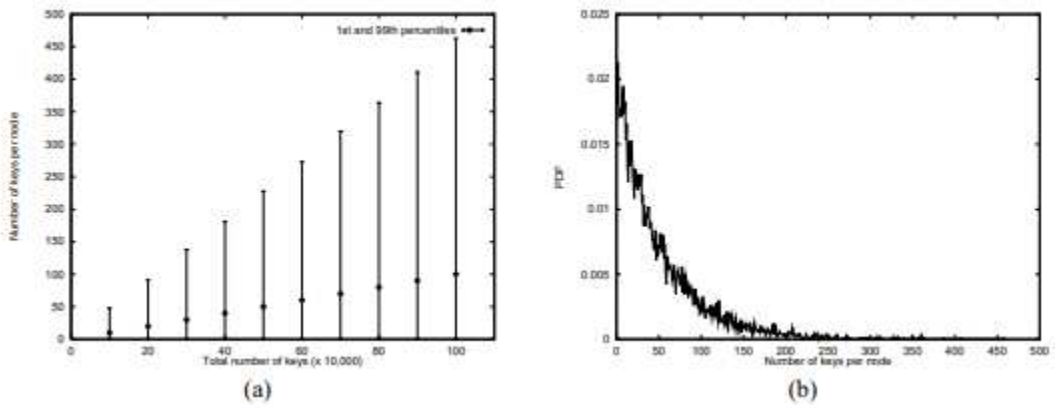
شناسه را فراهم می کند. برای به عنوان مثال، اگر ما logN گره مجازی به صورت تصادفی انتخاب شده را به هر گره تخصیص دهیم، با احتمال بالا، هر N سطل شامل O (logN) گره مجازی می شود.

به منظور بررسی این فرضیه، ما یک آزمایش را انجام می دهیم که در آن ما r گره مجازی را به هر یک از گره های واقعی تخصیص می دهیم. در این حالت، کلیدها به جای گره های واقعی با گره های مجازی مرتبط می شوند. ما دوباره یک شبکه با 104 گره واقعی و 106 کلید را در نظر می گیریم. شکل 9 اولین و 99 صدک را برای r برابر 1، 2، 5، 10، و 20 نشان می دهد. همانطور که انتظار می رود، 99امین صدک کاهش می یابد، در حالی که 1 صدک با تعداد گره های مجازی، r، افزایش می یابد. به خصوص، 99امین صدک از 04:08؟ تا 1:6 مقدار متوسط کاهش می یابد، در حالی که از اولین صدک از 0 تا 0:05 مقدار متوسط افزایش می یابد. بنابراین، اضافه کردن گره های مجازی به عنوان یک لایه غیر مستقیم می تواند به طرز قابل ملاحظه تعادل بار را بهبود بخشد. موازنه این است که هر گره واقعی در حال حاضر نیاز به r برابر فضای زیادی برای ذخیره جدول انگشت برای گره های مجازی آن دارد.

ما مشاهدات متعددی را با توجه به پیچیدگی متحمل شده توسط این طرح انجام دادیم. اول، مقدار تقریبی طول مسیر پرس و جو ، که در حال حاضر تبدیل به O (log(N log N)) = O (log N) می شود، تحت تاثیر قرار نگرفته است. دوم، فضای شناسه کلی تحت پوشش توسط گره های مجازی نگاشته شده روی همان گره واقعی با احتمال زیاد، کسر O (1 = N) از کل است که همان متوسط در شرایطی غیاب گره های مجازی است. از آنجا که تعداد پرس و جوهای به کار گرفته شده توسط یک گره تقریبا متناسب با فضای شناسه کل تحت پوشش آن گره است، تعداد بدترین حالت پرس و جوهای به کار گرفته شده توسط یک گره تغییر نمی کند. سوم، در حالی که حالت مسیریابی حفظ شده توسط یک گره در حال حاضر O (log2 N) است، این مقدار هنوز هم در عمل منطقی است؛ برای $N = 10^6, \log^2 N$ تنها 400 می باشد. در نهایت، در حالی که تعداد پیام های کنترل آغاز شده توسط یک گره با فاکتور عامل O (logN) را افزایش می یابد، تعداد مجانبی پیام های شاهد

دریافت شده از گره های دیگر تحت تاثیر نیست. برای دیدن چرایی این مورد، توجه داشته باشید که در صورت عدم وجود گره مجازی، با احتمال "معقول"، یک گره واقعی مسئول O (logN = N) فضای شناسه است. از آنجا که O (N logN) انگشت در سیستم کل وجود دارد، تعداد انگشتانی که به یک گره واقعی اشاره دارد، O (log2 N) است. در مقابل، اگر هر گره واقعی logN گره های مجازی را نگاشت کند، با احتمال بالا هر گره واقعی مسئول O (1 = N) فضای شناسه است. چون $O(N \log^2 N)$ انگشت در کل سیستم وجود دارد، با احتمال بالا، تعداد انگشتانی که به گره های مجازی نگاشت شده روی همان گره واقعی اشاره دارند هنوز هم O (log2 N) است.

فضای شناسه تحت پوشش یک گره مجازی نشان دهنده فاصله بین شناسه گره و شناسه پیش نیاز آن است. فضای شناسه پوشش داده شده توسط یک گره واقعی مجموع فضاهای شناسه تحت پوشش گره های مجازی آن است.



شکل 8. (الف) متوسط و اولین و 99امین صدک از تعداد کلیدهای ذخیره شده در هر گره در شبکه با 104 گره ذخیره می شود. (ب) تابع چگالی احتمال (PDF) برای تعداد کلیدها در هر گره. تعداد کلیدها 5×10^5. است.

C. طول مسیر

عملکرد وتر را تا حدی وابسته به تعداد گره هایی است که باید برای حل یک پرس و جو بازدید شود. از قضیه IV.2، با احتمال بالا ، این تعداد O (logN) است، که در آن N است که کل تعداد گره ها در شبکه است.

برای درک عملکرد مسیریابی وتر در عمل، ما یک شبکه با N = گره K2، با ذخیره سازی 100؟ کلید K2 در کل را شبیه سازی نمودیم. ما k را از 3 تا 14 تغییر دادیم و یک آزمایش جداگانه برای هر مقدار انجام دادیم. هر گره در یک آزمایش مجموعه ای تصادفی از کلیدها را برای پرس و جو از این سیستم برداشت، و ما طول مسیر هر پرس و جو را اندازه گیری نمودیم.

شکل 10 (a) نمودار متوسط، و اولین و 99امین صدک طول مسیر به عنوان تابعی از k است. همانطور که انتظار می رود، طول متوسط مسیر با تعداد گره ها به صورت لگاریتمی افزایش می یابد، همانند اولین و 99امین صدک. شکل 10 (b) نمودار PDF طول مسیر برای یک شبکه با 212 گره (K = 12) است.

شکل 10 (a) نشان می دهد که طول مسیر در حدود 1 2 log2 N است. مقدار ثابت (1/2) می تواند به صورت زیر قابل درک باشد. یک گره را در نظر بگیرید که یک پرس و جو را برای کلید به طور تصادفی انتخاب شده ایجاد می کند. فاصله در فضای شناسه بین گره و کلید را به صورت دودویی نشان دهید. مهم ترین (i ام) بیت این مسافت می تواند توان توسط 0 با پیروی از انگشت i ام گره اصلاح شود. اگر بیت قابل توجه بعدی فاصله 1 باشد، باید توسط یک انگشت اصلاح شود، اما اگر 0 باشد، آنگاه $i - 1^{st}$ انگشت دنبال نمی شود- درعوض، ما نیاز به دنبال نمودن تعداد یک ها در نمایش دودویی فاصله از گره به پرس و جو داریم. از آنجا که شناسه های گره بصورت تصادفی توزیع شده اند، ما انتظار اریم نیمی از بیت ها یک باشند. همانطور که در قضیه IV.2 مورد بحث قرار گرفت، پس از اینکه قابل توجه ترین بیت logN ثابت شود، در انتظار تنها یک گره بین موقعیت جریان باقی مانده و کلید وجود دارد. بنابراین در طول متوسط مسیر حدود $\frac{1}{2}\log_2 N$ خواهد بود.

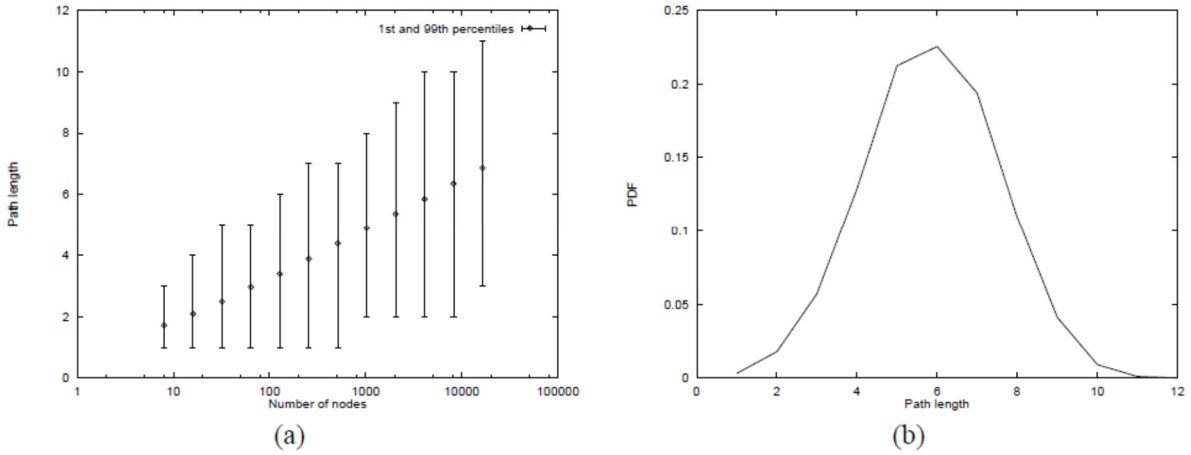
D.وقوع خرابی های همزمان گره

در این آزمایش، ما تاثیر وقوع خرابی بزرگ در عملکرد وتر و توانایی آن برای انجام صحیح مراجعات را ارزیابی نمودیم. ما یک شبکه با N = 1000 گره، که در آن هر گره یک لیست جانشین با اندازه $r = 20 = 2\log_2 N$ را حفظ می کند، در نظر می گیریم (نگاه کنید به بخش IV-E.3 برای بحث و گفتگو در موردانداره جانشین لیست). هنگامی که شبکه پایدار می شود، هر گره با p احتمال ناموفق می شود. پس از اینکه وقوع خرابی رخ می دهد، ما 10000 مراجعات تصادفی را انجام می دهیم. برای هر مراجعه، شماره وقفه های تجربه شده توسط مراجعه را ثبت می کنیم، تعداد گره های تماس یافته در مراجعه (از جمله تلاش ها برای تماس با گره های ناموفق)، و اینکه آیا مراجعه جانشین فعلی واقعی کلید را یافته است یا خیر. یک وقفه زمان اتفاق می افتد که یک گره برای تماس با گره ناموفق تلاش می کند. تعداد وقفه تجربه شده توسط یک مراجعه برابر با تعداد گره های ناموفق مواجه شده توسط عمل مراجعه است. برای تمرکز بر ارزیابی عملکرد وتر بلافاصله پس از وقوع خرابی، قبل از اینکه فرصتی برای اصلاح جداول مربوطه به خود داشته باشد، این آزمایشات ثبات را درست قبل از اینکه وقوع خرابی رخ دهد متوقف می کند و انگشتان اشاره به گره های ناموفق را از جداول انگشت حذف می کند. بنابراین گره های ناموفق تنها زمانی شناسایی می شوند که آنها در مدت پروتکل مراجعه موفق به پاسخ نشوند.

جدول II متوسط، اولین و 99امین صدک طول مسیر برای 000،10 مراجعه را پس از اینکه وقوع خرابی به عنوان یک تابع از p، بخشی از گره های ناموفق رخ می دهد، نشان می دهد. همانطور که انتظار می رفت، طول مسیر و تعداد وقفه زمانی که کسری از گره ها که ناموفق بوده اند، افزایش می یابد، زیاد می شوند.

برای تفسیر بهتر این نتایج، ما میانگین طول مسیر مراجعه را زمانی که هر گره دارای یک لیست جانشین با اندازه r است، برآورد می کنیم. با استدلال مشابه به یک مورد استفاده در بخش VC، فهرست جانشین با اندازه r

آخرین $\frac{1}{2}\log_2 r$ هاپ را از مسیر مراجعه به طور متوسط حذف می کند. میانگین طول مسیر مراجعه $\frac{1}{2}\log_2 N - \frac{1}{2}\log_2 r + 1$ است. آخرین عبارت (1) دسترسی به پیش نیاز کلید پرس و جو شده را زمانی که این پیش نیاز در فهرست جانشین گره قبلی یافت می شود، در نظر می گیرد. برای N=1,000 و r=20، میانگین طول مسیر 3:82 است که بسیار نزدیک به مقدار 3:84 در جدول دوم برای P = 0 نشان داده شده است



شکل 10. (الف) طول مسیر به عنوان تابعی از اندازه شبکه. (ب) PDF طول مسیر در مورد یک شبکه با 212 گره.

تعداد میانگین وقفه ها (اولین و 99امین صدک)

میانگین طول مسیر (اولین و 99امین صدک)

کسر گره های دارای وقوع خرابی

Fraction of failed nodes	Mean path length (1st, 99th percentiles)	Mean num. of timeouts (1st, 99th percentiles)
0	3.84 (2, 5)	0.0 (0, 0)
0.1	4.03 (2, 6)	0.60 (0, 2)
0.2	4.22 (2, 6)	1.17 (0, 3)
0.3	4.44 (2, 6)	2.02 (0, 5)
0.4	4.69 (2, 7)	3.23 (0, 8)
0.5	5.09 (3, 8)	5.10 (0, 11)

جدول II: طول مسیر و تعداد وقفه مراجعه تجربه شده به عنوان تابع کسری از گره هایی که به طور همزمان ناموفق هستند. اولین و 99امین صدک در داخل پرانتز هستند. در ابتدا، این شبکه دارای 1000 گره است.

در نظر بگیرید x پیشرفت صورت گرفته در فضای شناسه نسبت به یک کلید هدف مورد نظر در طول تکرار مراجعه خاص باشد، هنگامی که هیچ خرابی در سیستم وجود ندارد. سپس، فرض کنید که هر گره به طور مستقل با احتمال pبا وقوع خرابی مواجه می شود. همانطور که در بخش IV-E.3 مورد بحث قرار گرفته است، در هر تکرار مراجعه، هر گره بزرگترین انگشت زنده را انتخاب می کند (از جدول انگشتان خود) که قبل از کلید هدف می آید. بنابراین پیشرفت صورت گرفته در همان تکرار مراجعه در فضای شناسه x با احتمال (1-p)، تقریبا X / 2 با احتمال p*(1-p) تقریبا $x/2^2$ با احتمال و غیره است. پیشرفت مورد انتظار صورت گرفته به سمت کلید برابر $\sum_{i=0}^{\infty} \frac{x}{2^i}(1-p)p^i = x(1-p)/(1-p/2)$ است. به عنوان یک نتیجه، طول میانگین مسیر تقریبا $\frac{1}{2}\log_d N - \frac{1}{2}\log_d r + 1$ می شود که در ان $d = 1.7 = \log_2\left(\frac{1-p/2}{1-p}\right)$. به عنوان یک مثال، طول مسیر میانگین برای p=0.5 برابر 5.76 است. یک دلیل که برای آن مقدار پیش بینی شده بزرگتر از مقدار اندازه گیری شده است در جدول II وجود دارد، زیرا سری استفاده شده برای ارزیابی d در عمل متنهای است. این کار منجر به تخمین مقدار d می شود که به نوبه خود منجر به تخمین طول مسیر میانگین می شود.

حل، توجه خود را به تعداد وقفه ها معطوف کنید. در نظر بگیرید که m تعداد میانگین گره های تماس یافته در مدت عملیات مراجعه باشد. تعداد مورد انتظار وقفه های تجربه شده در مدت عملیات مراجعه m*p است و طول مسیر میانگین l = m * (1-p) است. با توجه به طول مسیر میانگین در جدول II، تعداد مورد انتظار وقفه ها برای p=0.1 برابر 0.45، برای 1.06، برابر p=0.2، 1.90 برای p=0.3، 3.13 برای p=0.4، و 5.06 برای p=0.5 است. این مقادیر به خوبی را مقادیر اندازه گیری شده وقفه های نشان داده شده در جدول II تطبیق یافته اند.

نهایتاً ما یادآوری می کنیم که در شبیه سازی های ما، تمام مراجعات به طور موفقیت آمیز حل و فصل می شوند که از ادعای استحکام قضیه IV.5 حمایت می کند.

E.مراجعات در مدت پایداری

در آین آزمایش، ما عملکرد و دقت مراجعات وتر را زمانی که گره ها به طور مداوم وصل می شوند و ترک می کنند را ارزیابی می کنیم. رویه ترک از بهینه سازی های ترسیم شده در بخش IV.E.4 استفاده می کند. مراجعات کلیدی مطابق با فرآیند پواسن در نرخ یکی در هر ثانیه تولید می شوند. پیوستن و ترک های داوطلبانه توسط فرآیند پواسن با میانگین ورودی با نرخ R مدلسازی می شوند. هر گره روال پایدارسازی را در بازه هایی اجرا می کند که به طور یکنواخت در بازه [15,45] ثانیه توزیع شده است؛ به خاطر داشته باشید که تنها جانشین و ورودی جدول انگشت برای هر تماس پایدارسازی می شوند، بنابراین بازه مورد انتظار بین پایدارسازی های متوالی یک ورودی جدول انگشت معین بسیار طویل تر از پایدارسازی میانگین دوره 30 ثانیه است. این شبکه با 1000 گره آغار می شود و هر گره دوباره فهرست جانشین با اندازه r = 20 = 2logN را حفظ می کند. توجه کنید که حتی اگر هیچ وقوع خرابی وجود نداشته باشد، وقفه ها ممکن است هنوز در مدت عملیات مراجعه رخ دهند؛ یک گره که برای تماس با یک انگشت تلاش می کند دارای وقفه خواهد بود.

جدول III نشاندهنده میانگین ها و اولین و 90امین صدک طول مسیر و تعداد وقفه های تجربه شده توسط عملیات مراجعه به عنوان تابعی از نرخ R است که در آن گره ها می پیوندند و ترک می کنند. نرخ R=0.05 متناظر با یک گره پیوسته و ترک کننده هر 20 ثانیه به طور متوسط است. برای مقایسه، به خاطر داشته باشید که هر گره پروتکل پایداری را هر 30 ثانیه درخواست می کند. بنابراین، R در نرخ یک پیوند و ترک در هر 1.5 ثانیه دوره پایداری تا نرخ 12 پیوند و 12 ترک در هر دوره پایداری گستره دارد.

همانطور که در بخش V-D بحث شد، طول مسیر میانگین در حالت پایدار حدود $\frac{1}{2}\log_2 N - \frac{1}{2}\log_2 r + 1$ است. دوباره، چون N=1000 و r=20 است، طول مسیر میانگی 3.82 است. همانطور که در جدول III نشان داده شد، طول اندازه گیری شده مسیر بسیار به این مقدار نزدیک است و زمانی که R افزایش یابد تغییر می کند. دلیل این مورد اینست که تعداد وقفه های تجربه شده توسط یک مراجعه نسبتاً کم است و بنابراین دارای اثر مینیمم روی طول مسیر است. از طرف دیگر، تعداد وقفه ها با R افزایش می یابد. برای درک این نتیجه، این شبکه تقریباً دارای 1000 گره است.

نرخ پیوستن/ترک گره (در هر ثانیه// در هر دوره پایدارسازی)	طول مسیر میانگین (اولین و 99امین صدک)	تعداد میانگین وقفه ها (اولین و 99امین صدک)	وقوع خرابی های مراجعه (در هر 10000 مراجعه)
0.05 / 1.5	3.90 (1, 9)	0.05 (0, 2)	0
0.10 / 3	3.83 (1, 9)	0.11 (0, 2)	0
0.15 / 4.5	3.84 (1, 9)	0.16 (0, 2)	2
0.20 / 6	3.81 (1, 9)	0.23 (0, 3)	5
0.25 / 7.5	3.83 (1, 9)	0.30 (0, 3)	6
0.30 / 9	3.91 (1, 9)	0.34 (0, 4)	8
0.35 / 10.5	3.94 (1, 10)	0.42 (0, 4)	16
0.40 / 12	4.06 (1, 10)	0.46 (0, 5)	15

جدول III: طول مسیر و تعداد وقفه تجربه مراجعه به عنوان تابعی از نرخ پیوستن و ترک گره. اولین و 99امین صدک در پرانتز

اجازه دهید اشاره گر انگشت خاص f را از گره n در نظر بگیریم و کسر پیمایش های مراجعات آن انگشت که با وقفه مواجه می شود (توسط تقارن، این مورد برای تمام انگشتان یکسان خواهد بود. از دیدگاه آن انگشت، این تاریخچه از ترک سه نوع رخداد ساخته می شود: (1) پایدارسازی های آن انگشت (2) انحرافات از گره اشاره شده به آن توسط انگشت و (3) مراجعاتی که انگشت را پیمایش می کند. یک مراجعه در صورتی سبب یک وقفه می شود که انگشت به گره حرکت کرده اشاره کند. این مورد دقیقاً زمانی رخ می دهد که این رخداد که فوراً پیش از مراجعه بوده، یک حرکت باشد- اگر رخداد قبلی یک پایدارسازی باشد، آنگاه این گره در حال حاضر زنده است؛ به طور مشابه، اگر رخداد قبلی یک مراجعه باشد، آنگاه آن مراجعه یک اخراج سببی آن اشاره گر انگشت مرده را متوقف می کند. بنابراین ما نیاز به تعیین کسر رخدادهای مراجعه در تاریخی داریم که فوراً توسط رخداد انحراف مقدم تر می شود.

برای ساده سازی تحلیل فرض می کنیم که مانند اتصالات و ترک ها، پایداری سازی مطابق با فرآیند پواسن اجرا می شود. تاریخچه ما یک جاگذاری از سه فرایند پواسن است. گره انگشتی به عنوان فرآیند پواسن در نرخ R’ = R/N منحرف می شود. پایدارسازی آن انگشت در نرخ S رخ می دهد (و چنین انحرافی را آشکار می کند). در هر دور پایدارسازی، یک گره گره دیگر را در جدول انگشت خود یا یک گره در فهرست جانشین خود پایدار می کند (3 log N مانند گره ها در مورد ما وجود دارد). چون عملیات پایدارسازی به یک عملیات مراجعه کاهش می یابد (شکل 6 را ببینید)، هر عملیات پایدارسازی از l انگشت به طور متوسط استفاده می کند که در آن l طول میانگین مسیر مراجعه است. به عنوان یک نتیجه، نرخی که در آن یک انگشت توسط عملیات پایدارسازی لمس می شود $S = (1/30) * l/(3 \log N)$ است که در آن 1/30 نرخ متوسطی است که در آن هر گره پایدارسازی را درخواست می کند. نهایتاً، مراجعاتی که از آن انگشت استفاده می کنند نیز یک فرآیند پواسن هستند. به خاطر داشته باشید که این مراجعات (به طور کلی) به عنوان فرآیند پواسن با نرخ یک مراجعه در هر ثانیه تولید می شوند. چنین مراجعه ای از l انگشت به طور متوسط استفاده می کند در حالیکه N log N

انگشت به طور کلی وجود دارد. بنابراین یک انگشت خاص با احتمال l/ (N log N) استفاده می شود، که بدین معنی است که این انگشت مطابق با فرآیند پواسن در نرخ L = l / (N log N) استفاده شده است.

ما سه فرآیند جاگذاری شده پواسن داریم (مراجعات، انحرافات و پایدارسازی ها). چنین اتحادی از فرآیندهای پواسن به خودی خود یک فرآیند پواسن با نرخ برابر با مجموع سه نرخ مورد نظر است. هر بار که یک رخداد در این فرآیند اتحاد رخ می دهد، به یکی از سه فرآیند مورد نظر با احتمال متناسب با نرخ های آن فرآیندها منصوب می شود. به بیانی دیگر، تاریخچه دیده شده توسط یک گره مانند یک ترتیب تصادفی به نظر می رسد که در آن هر رخداد یک انحراف با احتمال زیر است

$$p_t = \frac{R'}{R'+S+L} = \frac{\frac{R}{N}}{\frac{R}{N}+\frac{l}{90\log N}+\frac{l}{N\log N}} = \frac{R}{R+\frac{l*N}{90\log N}+\frac{l}{\log N}}.$$

به طور خاص، این رخداد که فوراً قبل از هر مراجعه قرار دارد، یک انحراف با این احتمال است. این احتمال که مراجعه با وقفه مواجه می شود. نهایتاً، تعداد مورد انتظار وقفه های تجربه شده توسط یک عملیات مراجعه برابر $R/(R/l+N/(90\log N)+1/\log(N))$است. به عنوان مثال، تعداد مورد انتظار وقفه ها 0.041 برای R=0.05 و 0.31 برای R=0.4است. این مقادیر نزدیک به مقادیر اندازه گیری شده در جدول III معقول هستند.

در ستون آخر در جدول III تعداد وقوع خرابی ها در هر 10000 مراجعه نشان داده شده است. دلیل این وقوع خرابی ها، ناسازگاری حالت است. به خصوص، بر خلاف این حقیقت که هر گره یک فهرست جانشین با 2 log N را حفظ می کند، این امکان وجود دارد که برای دوره های کوتاه زمانی، یک گره بتواند به جانشین نادرست اشاره کند. فرض کنید که در زمان t، گره n اولین و دومین جانشین خود را می شناسد، s1 و s2. فرض کنید

که درست بعد از زمان t، یک گره جدید s بین s1, s2 به شبکه می پیوندند و اینکه s1 قبل از اینکه n دارای فرصتی برای کشف s باشد، ترک می کند. زمانی که n می فهمد که s1 ترک کرده است، n آن را با s2 جایگزین می کند، نزدیک ترین جانشین n آن را می شناسد. به عنوان یک نتیجه، برای هر کلید $id \in (n, s)$، n گره s2 به جای s باز می گردد. هرچند، زمان بعدی n پایدارسازی برای s2 را درخواست می کند، n جانشین صحیح خود s را یاد می گیرد.

F. بهبود نهفتگی مسیریابی

در حالیکه وتر اطمینان می بخشد که طول متوسط مسیر تنها ½ log 2 N است، نهفتگی مراجعه می تواند نسبتاً بزرگ باشد. دلیل این مطلب اینست که شناسه های گره به طور تصادفی توزیع می شوند و بنابراین گره های نزدیک به فضای شناسه می توانند دور از شبکه قرار گیرند. در کار قبلی [8]، ما برای کاهش کمبود مراجعه با الحاق ساده پروتکل وتر تلاش می کنیم که تنها اطلاعات قبلی را در جدول انگشت گره به کار می گیرد. این ایده برای انتخاب انگشت هاپ بعدی مبتنی بر پیشرفت در فضای شناسه و نهفتگی در شبکه مورد نظر است که برای ماکزیمم نمودن قبلی تلاش می کند در حالیکه مورد دوم را مینیمم می کند.

کشش (10امین، 90امین صدک)

تعداد جانشین های انگشت	تکراری		بازگشتی	
	3-d space	Transit stub	3-d space	Transit stub
1	7.8 (4.4, 19.8)	7.2 (4.4, 36.0)	4.5 (2.5, 11.5)	4.1 (2.7, 24.0)
2	7.2 (3.8, 18.0)	7.1 (4.2, 33.6)	3.5 (2.0, 8.7)	3.6 (2.3, 17.0)
4	6.1 (3.1, 15.3)	6.4 (3.2, 30.6)	2.7 (1.6, 6.4)	2.8 (1.8, 12.7)
8	4.7 (2.4, 11.8)	4.9 (1.9, 19.0)	2.1 (1.4, 4.7)	2.0 (1.4, 8.9)
16	3.4 (1.9, 8.4)	2.2 (1.7, 7.4)	1.7 (1.2, 3.5)	1.5 (1.3, 4.0)

جدول IV: کشش نفهتگی مراجعه برای یک سیستم وتر با 2^16 گره زمانی که یک مراجعه در سبک تکراری و بازگشتی انجام می شود. دو مدل شبکه در نظر گرفته می شوند: یک فضای اقلیدسی سه بعدی، و شبکه استاب گذری

در حالیکه این الحاق پروتکل برای پیاده سازی ساده است و نیاز به هر حالت اضافی ندارد، عملکرد آن در تحلیل مشکل است [8]. در این بخش، ما یک الحاق پروتکل جایگزین را ارائه می دهیم که ارائه دهنده عملکرد بهتر در هزینه افزایش مختصر حالت وتر و پیچیدگی پیام است. ما تاکید می کنیم که به طور فعال در حال بررسی تکنیک هایی برای مینیمم نمودن نهفتگی مراجعه هستیم و بهبودهای بیشتری را در آینده انتظار داریم.

ایده اصلی طرح ما، نگهداری مجموعه ای از گره های جایگزین برای هر انگشت است (یعنی، گره ها با شناسه های مشابه که تقریباًبرای مقاصد مسیریابی مناسب هستند) و بنابراین پرس و جوها را با انتخاب نزدیک ترین گره در میان گره های جایگزین مطابق با برخی مقیاس های نزدیکی شبکه مسیریابی می کنند. به خصوص، هر گره با هر انگشت خود، فهرستی از جانشین های بلافاصله f ارتباط دارد. به علاوه، ما تابع find_succesor را در شکل 5 متناظراً اصلاح می کنیم: به جای بازگرداندن ساده بزرگترین انگشت، f، که قبل از ID پرس و جو شده است، تابعی که نزدیک ترین گره (برحسب فاصله شبکه بندی) در میان f و جانشین های s خود باز می گرداند. برای سادگی، s=r، که در آن r طول فهرست جانشین است؛ می توان الزامات ذخیره را برای جدول مسیریابی با نگهداری برای هر انگشت f، تنها گره نزدیک n در میان جانشین های s کاهش داد. برای به روزسازی n، یک گره میتواند به سادگی از f فهرست جانشین آن را درخواست کند. این گره می تواند n را به صورت متناوبی به روزسازی نماید یا زمانی که آشکار می کند که n ناموفق بوده است. مشاهده کنید که این ابتکار می تواند تنها در پیاده سازی بازگشتی (نه تکراری) مراجعه اعمال نماید زمانی که گره پرس و جوکننده هیچ اندازه گیری فاصله ای برای انگشتان در هر گره مسیر ندارد.

برای توضیح بازده این ابتکار، ما یک سیستم وتر را با 2^16 گره و دو توپولوژی شبکه در نظر می گیریم:

- فضای سه بعدی: فاصله شبکه به عنوان فاصله مقیاس در فضای سه بعدی مدلسازی می شود. این مدل توسط تحقیقات اخیر [19] تحریک می شود که نشان دهنده اینست که نهفتگی شبکه بین دو گره در

اینترنت می تواند به عنوان فاصله هندسی (با دقت خوب) در یک فضای اقلیدسی d بعدی که در آن d≥3 است، مدلسازی شود.

- استاب گذار: یک توپولوژی استاب گذار با 5000 گره که در آن نهفتگی های لینک 50 میلی ثانیه برای لینک های حوزه گذار-داخلی هستند، 20 میلی ثانیه برای لینک های استاب گذار و 1 میلی ثانیه برای لینک های حوزه استاب داخلی است. گره های وتر به طور تصادفی به گره های استاب منصوب می شوند. این توپولوژی شبکه دارای هدف انعکاس سازماندهی سلسله مراتبی اینترنت امروزی است.

ما از کشش مراجعه به عنوان مقیاس اصلی برای ارزیابی ابتکار خود استفاده می کنیم. کشش مراجعه به صورت نسبت بین (1) نهفتگی مراجعه وتر از زمانی که مراجعه برای زمانی آغاز می شود که تعریف می شود که نتیجه به راه انداز بازگردانده می شود و (2) نهفتگی مراجعه بهینه با استفاده از شبکه مورد نظر. مورد دوم به صورت زمان دو سفره بین راه انداز و مسئول سرور برای ID پرس و جو شده محاسبه می شود.

جدول IV نشاندهنده میانه، 10امین و 99امین صدک کشش مراجعه در 10000 مراجعه برای سبک های تکراری و بازگشتی است. این کشش زمانی که s از یک تا 16 افزایش می یابد به طور چشمگیری کاهش می یابد.

همانطور که انتظار می رود، این نتایج نیز اثبات می کنند که مراجعات بازگشتی سریع تر از مراجعات تکراری اجرا می شوند. بدون هر بهینه سازی، سبک مراجعه بازگشتی تقریباً دو برابر سریع تر از سبک تکراری انتظار می رود: یک مراجعه تکراری متحمل یک رفت و برگشت در هر هاپ می شود در حالیکه مراجعه بازگشتی متحمل یک کمبود تک راهه می شود.

توجه کنید که در یک فضای اقلیدسی 3 بعدی، فاصله مورد انتظار از یک گره به نزدیک ترین گره از مجموعه s+1 گره تصادفی متناسب با $(s+1)^{1/3}$ است. از اینرو تعداد هاپ های وتر زمانی که s افزایش می یابد

تغییر نمی کند، ما انتظار داریم که نهفتگی مراجعه متناسب با $(s+1)^{1/3}$ باشد. این مشاهده با نتایج ارائه شده در جدول IV سازگار است. برای نمونه در s =16، داریم $17^{1/3} = 2.57,$ که نزدیک به کاهش مشاهده شده مقدار میانه برای کشش مراجعه از s=1 تا s=16 است.

VI. کارهای آینده

کارهای باقیمانده که باید انجام شود، بهبود قابلیت ارتجاع وتر در مقابل پارتیشن های شبکه و گره های رقیب و بازده آن است.

وتر می تواند پارتیشن هایی را آشکار کند که گره های آنها یکدیگر را می شناسند. یک روش برای دستیابی به این دانش برای هر گره، شناختن مجموعه ای کوچک از گره های اولیه است. دیگر روش می تواند برای گره های به منظور حفظ حافظه طولانی مدت مجموعه تصادفی گره هایی باشد که در گذشته مواجه شده اند؛ اگر یک پارتیشن تشکیل شود، مجموعه های تصادفی در یک پارتیشن احتمالاً شامل گره هایی از دیگر پارتیشن می شوند.

یک مجموعه دارای اشکال یا خراب برای شرکت کنندگان وتر می تواند نشاندهنده دیدگاهی نادرست از حلقه وترباشد. با فرض اینکه وتر داده ها برای موقعیت یابی استفاده می شود، از لحاظ رمزنگاری تایید شده است، این تهدیدی برای دردسترس بودن داده ها است نه تایید آنها. یک روش برای کنترل سازگاری کلی برای هر گره n درخواست متناوب انجام مراجعه وتر برای n است؛ اگر مراجعه نشاندهنده گره n نباشد، این کار می تواند نمایشگر قربانی هایی باشد که دیدی سازگار از حلقه وتر را نمی بینند.

حتی ½ log 2 N پیام در هر مراجعه می تواند برای برخی کاربردهای وتر زیاد باشد، به خصوص اگر هر پیام به میزبان اینترنت تصادفی فرستاده شود. به جای جایگذاری انگشتان آن در فواصل که همه دارای توان 2 هستند، وتر می تواند به آسانی برای جایگذاری انگشتان خود در فواصلی تغییر یابد که همه دارای توان 1/d + 1

هستند. تحت چنین طرحی، یک هاپ مسیریابی تک می تواند فاصله را برای پرس و جو به 1/(1+d) برابر فاصله اصلی کاهش دهد که بدین معنی است که $\log_{1+d} N$ هاپ کافی است. هرچند، تعداد انگشتان مورد نیاز، به $\log N/(\log(1+1/d) \approx O(d \log N).$ افزایش می یابد.

VII. نتیجه گیری

بسیار یاز کاربردهای نظیر به نظیر توزیع شده نیاز به تعیین گره ای دارند که ایتم داده را ذخیره می کند. پروتکل وتر این مسئله چالش ساز را با روشی غیرمتمرکز حل می کند. این روش، مقدار اولیه قوی را ارائه می دهد، با توجه به یک کلید، گره مسئول برای ذخیره مقدار کلید را ارائه می دهد و به طور کارامدی این کار را انجام می دهد. در حالت پایدار، شبکه N گره، هر گره اطلاعات مسیریابی را برای تنها O (log N) گره دیگر حفظ می کند و تمام مراجعات را توسط O (log N) پیام برای دیگر گره ها حل و فصل می کند.

ویژگی های جذاب وتر شامل سادگی آن، تصحیح متحمل و عملکرد محتمل حتی در مواجهه با انحرافات ورودی های گره جاری می شود. این کار برای انجام وظیفه صحیح ادامه می یابد، البته عملکرد تنزل یافته، زمانی که اطلاعات گره تنها به طور جزئی صحیح است. تحلیل نظری ما، و نتایج شبیه سازی تایید می کند که وتر به خوبی با تعداد گره ها مقیاس بندی می شود، از تعداد زیادی از وقوع خرابی ها و اتصالات همزمان بازیابی می شود و بیشتر مراجعات را حتی در مدت بازیابی پاسخگویی می نماید.

ما باور داریم که وتر یک جز باارزش برای کاربردهای توزیع شده مقیاس بزرگ، نظیر به نظیر مانند اشتراگ فایل همکارانه، سیستم های ذخیره دردسترس با اشتراک زمانی، شاخص های توزیع شده برای اسناد و کشف خدمات و پلت فرم های محاسبه توزیع شده با مقیاس بزرگ است. تجربه اولیه ما با وتر بسیار نویدبخش است. ما قبلاً چندین کاربرد نظیر به نظیر را با استفاده از وتر، شامل کاربرد به اشتراک گذاری فایل ساخته ایم [9]. این نرم افزار در http://pdos.lcs.mit.edu/وتر/ دردسترس است.