نظرة عامة على الريديس الجزء الثاني

تحدثنا في الجزء الأول من السلسلة التعريفية بالريديس عن مبدأ عمله وميزاته، نتابع معكم في هذا الجزء الحديث عن بعض الفروقات الأساسية بينه وبين المخازن الأخرى والمزيد من الميزات والأسئلة الشائعة حولها.

ما الفرق بين ريديس ومخازن المفتاح-القيمة key-value ؟

حسناً يمكن القول إن هناك فرقان أساسيان:

1. يأخذ ريديس منحنى تطوري مختلف بالنسبة لقواعد بيانات المفتاح-القيمة، حيث يمكن أن تحتوي القيم على أنماط أكثر تعقيداً

من البيانات، مع عمليات ذرية معرفة لهذه الأنماط من البيانات. ترتبط أنماط البيانات الخاصة بريديس بشكل كبير ببنى البيانات **data structures** الأساسية وهي معروفة للمبرمجين كذلك، من دون طبقات تجريدية إضافية.

2. ريديس عبارة عن قاعدة بيانات في الذاكرة **in-memoryy** ولكن ثابتة على قاعدة بيانات القرص الصلب **disk database**، لذلك تقدم مقايضات مختلفة حيث تتحقق سرعات عالية للقراءة والكتابة مع محدودية مجموعات البيانات، التي لا يمكن أن تكون أكبر من الذاكرة. ميزة أخرى في قواعد البيانات في الذاكرة هو أن تمثيل الذاكرة لبنى البيانات المعقدة من السهل معالجته مقارنةً بنفس البنى على القرص، لذلك يستطيع ريديس القيام بالكثير، ولكن بتعقيد داخلي أقل.

وفي الوقت نفسه لا تحتاج صيغتا التخزين على القرص **RDB** و**AOFF** (المذكورتان سابقاً) لأنهما تكونان مناسبتين للوصول العشوائي، لذلك تكونان مضغوطتان وتوّلدان دائماً بطريقة الإلحاق فقط **append-only** (حتى تعاقب سجل **AOF** هو عملية إلحاق فقط، حيث تتولد النسخة الجديدة من نسخة البيانات في الذاكرة).

ما هي بصمة ذاكرة ريديس Redis memory footprint؟

إليك بعض الأمثلة (لأنماط التثبيت 64 بت)

- في حالة عدم الاستخدام تحتاج وحدة الريديس تقريباً 1 ميغابايت من الذاكرة.
- 1 مليون مفتاح صغير أزواج من قيم سلاسل محارف تحتاج تقريباً 100 ميغابايت من الذاكرة.
- 1 مليون مفتاح قيمة هاش **Hash value** تمثل كائن **object** بخمس حقول تحتاج تقريباً 200 ميغابايت من الذاكرة.

إن اختبار حالة الاستخدام أمر بسيط باستخدام أداة ريديس المرجعية **redis-benchmark** من أجل توليد مجموعات عشوائية من البيانات والتحقق من المساحة المستخدمة باستخدام أمر **INFO memory**.

تستخدم أنظمة 64 بت مساحة ذاكرة أكبر من أنظمة 32 بت من أجل تخزين المفاتيح نفسها، وخاصةً إذا كانت المفاتيح والقيم صغيرة. وذلك لأن المؤشرات **pointers** تأخذ 8 بايت في أنظمة 64 بت. أما الميزة فهي أنك تحصل على ذاكرة أكبر في أنظمة 64 بت، ولذلك من أجل تشغيل مخدمات ريديس كبيرة تكون أنظمة الـ 64 بت مطلوبة نوعاً ما.

أما البديل فهو شاردنغ **shardingg**. (شاردنغ: هي طريقة لتوزيع البيانات على آلات متعددة. أي تقسيم قاعدة بيانات كبيرة إلى أجزاء أصغر يسهل التعامل معها).

إنّ عمليات وميزات ريديس عالية المستوى تثير الإعجاب، ولكنها تأخذ كل شيء إلى الذاكرة ولا يمكن الحصول على مجموعة بيانات أكبر من الذاكرة. فهل هناك خطط لتغيير ذلك؟

قام مطورو ريديس في الماضي بتجريب الذاكرة الافتراضية وأنظمة أخرى من أجل السماح بمجموعة من البيانات أكبر من ذاكرة الوصول العشوائي **RAM**. وسنكون سعداء لو استطعنا تقديم بيانات من الذاكرة، وأقراص مستخدمة للتخزين. لا يوجد أي خطط إلى هذه اللحظة لإنشاء قرص نهاية خلفية **disk backendd** للريديس. على أي حال، الريديس هو عبارة عن نتيجة مباشرة لتصميمه الحالي.

إذا لم تكن مشكلتك مساحة ذاكرة الوصول العشوائي الكلية المطلوبة، وإنما حاجتك إلى تقسيم مجموعة بياناتك إلى أكثر من وحدة ريديس، فعليك قراءة المقال في الرابط التالي.

هل يعد استخدام الريديس مع قاعدة بيانات على القرص فكرة جيدة؟

أجل، يتضمن نمط تصميم شائع، أخذ بيانات حجمها صغير تُكتب بكثافة **write-heavy small dataa** في الريديس (وبيانات تحتاجها لكي تقوم بنى بيانات الريديس بنمذجة مشكلتك بشكل فعّال)، وأجزاء كبيرة من البيانات إلى **SQL** أو قاعدة بيانات ثابتة على القرص.

هل هناك أي شيء نستطيع فعله من أجل التقليل من استهلاك ريديس للذاكرة؟

إذا كان بإمكانك، استخدم ريديس على جهاز 32 بت. واستغل قيم الهاش الصغيرة ، واللوائح **listss**، والمجموعات المصنفة **sorted sets** ومجموعات الأعداد الصحيحة **integers** بشكل جيد، وذلك لأن بإمكان الريديس تمثيل أنماط البيانات هذه في الحالة الخاصة لبضعة عناصر بطريقة مضغوطة أكثر. وهناك المزيد من المعلومات حول ذلك في المقال التالي.

ما الذي يحصل عندما تمتلئ ذاكرة ريديس؟

سيتم إنهاء ريديس إما من قبل **OOM killer** لنواة اللينكس **Linux kernell**، وسينهار مع إظهار خطأ، أو سيصبح بطيء. في أنظمة التشغيل الحديثة، إعادة دالة **malloc** ()، للنتيجة **NULL** ليس بالأمر الشائع، عادةً يبدأ المخدم بالتبديل **swapping**، وبذلك سوف يسوء أداء ريديس، لذلك ستلاحظ في الغالب أن هناك خطب ما.

سيعطي الأمر **INFO** حجم الذاكرة الذي يستخدمه ريديس لكي تتمكن من كتابة نصوص **scripts** تراقب مخدّمات ريديس وتتحقق من وجود حالات حرجة.

يملك ريديس حماية داخلية **built-in protections** تسمح للمستخدم أن يضع حداً أقصى لاستخدام الذاكرة. باستخدام الخيار **maxmemory** في ملف التهيئة **config file** لوضع حداً للذاكرة التي يمكن لريديس استخدامها.

في حال تم الوصول إلى هذا الحد سيبدأ ريديس بالاستجابة لكتابة الأوامر بخطأ (ولكن لا يزال يسمح بأوامر القراءة فقط)، أو بإمكانك أن تقوم بالتهيئة لكي تطرد البيانات عند الوصول لأعلى حد من الذاكرة في حال كنت تستخدم ريديس للتخزين المؤقت **caching**.

في حال كنت تريد استخدام ريديس كذاكرة مؤقتة ومستخدمة مؤخراً **LRU cache** فعليك الاطلاع على هذا المستند.

لماذا يحدث الفشل في حفظ البيانات في الخلفية بظهور خطأ fork() في نظام اللينكس حتى مع استخدام الكثير من ذاكرة الوصول العشوائي الحرة؟

الجواب المختصر:

echo 1 > /proc/sys/vm/overcommit_memory

أما الجواب بالتفصيل:

يعتمد مخطط حفظ بيانات الخلفية في ريديس على عملية أخذ النسخ عند الكتابة **copy-on-writee** الدلالية للتشعب **fork** في أنظمة التشغيل الحديثة: يقوم التشعب بخلق عملية "ابن" **child process** وهي نسخة طبق الأصل عن العملية الأصلية "الأب" **parent**.

تلقي العملية الابن بقاعدة البيانات على القرص وتقوم بالخروج في النهاية. نظرياً يجب أن يستهلك الابن نفس مقدار الذاكرة التي يستخدمها الأب لكونه نسخة، ولكن بفضل عملية "أخذ النسخ عند الكتابة" الدلالية التي تطبق في معظم أنظمة التشغيل الحديثة، فسيتشارك الأب والابن صفحات الذاكرة المشتركة.

لن يتم تكرار الصفحة إلا إذا تغيرت في الابن أو الأب، وبما أنه يمكن لجميع الصفحات أن تتغير نظرياً فيما يتم حفظ العملية الابن، لا يمكن للينكس أن يعرف مسبقاً مقدار الذاكرة التي يحتلها الابن، لذلك إذا تم تعيين الإعداد **overcommit_memory** إلى قيمة صفر، فسيفشل التشعب إلا إذا كان هنالك ذاكرة وصول عشوائية كافية لتكرار جميع الصفحات الأب في الذاكرة. ما نخلص إليه هو أنه في حال كان لديك مجموعة بيانات ريديس حجمها 3 غيغابايت، وذاكرة وصول عشوائية حرة 2 غيغابايت فلن ينجح الأمر.

عند تعيين الإعداد **overcommit_memory** إلى قيمة 1 يتم اخبار نظام اللينكس بأن عليه أن يرتاح وأن ينفذ التشعب بطريقة تخصيص أكثر تفاؤلاً، وهذا ما نريده للريديس بالضبط.

هل لقطات ريديس على القرص Redis on-disk-snapshots ذرية؟

أجل، عملية الحفظ في الخلفية دائماً متشعبة عندما يكون الخادم خارج تنفيذ الأمر، لذلك كل أمر ذري في ذاكرة الوصول العشوائية يكون ذرياً أيضاً بالنسبة للَقطة القرص **disk snapshot**.

إنّ ريديس مفرد السلسلة single threaded. فكيف يمكن استغلال وحدات المعالجة المركزية CPU أو الأنوية cores المتعددة؟

من غير المرجح أن تضيّق عليك وحدة المعالجة المركزية عند استخدام الريديس، عادةً ما يكون الريديس مرتبط بالذاكرة أو الشبكة. على سبيل المثال، باستخدام المعالجة التفرعية **pipelining** يمكن لريديس يعمل على نظام لينكس عادي أن يلبي 500 ألف طلب في الثانية، لذلك إن كان تطبيقك يستخدم أوامر **O(N)** أو **O(log(N))**، فإنك لن تستخدم الكثير من قدرة المعالجة المركزية.

ولكن، من أجل زيادة استخدام وحدة المعالجة المركزية يمكنك البدء بوحدات متعددة للريديس في الجهاز نفسه، واستخدامها كمخدّمات مختلفة. في مرحلة ما لن يكون الجهاز الواحد كافياً، لذلك إن كنت تريد استخدام أكثر من وحدة معالجة مركزية فعليك التفكير بطريقة للتقسيم كما ذكر سابقاً.

يمكنك إيجاد المزيد من المعلومات المفيدة عن استخدام الوحدات المتعددة من الريديس هنا.

ما هو أقصى عدد للمفاتيح التي يمكن لوحدة ريديس أن تخزنها؟ وما هو أقصى عدد من العناصر في الهاش، اللائحة، المجموعة، والمجموعة المصنفة؟

يمكن لريديس أن يتعامل مع\(2^{32}\) مفتاح، وتم اختباره للتعامل مع مالا يقل عن 250 مليون مفتاح لوحدة ريديس. كل هاش، وقائمة، ومجموعة، ومجموعة مصنفة يمكن أن تخزن \(2^{32}\) عنصر.

بمعنى آخر، ما يقيدك هو مقدار الذاكرة المتوفرة على نظامك.

يدعي المخدم التابع أنه يملك عدد مختلف من المفاتيح مقارنة بالسيد، لماذا؟

إذا كنت تستخدم مفاتيح بزمن استخدام محدد فهذا الأمر طبيعي. إليك ما سيحدث:

- يوّلد السيد ملف قاعدة بيانات علائقية **RDB** في عملية المزامنة الأولى مع التابع.
- لن يتضمن ملف **RDB** مفاتيح منتهية مسبقاً في السيد، ولكن التي ما زالت موجودة في الذاكرة.
- لا تزال هذه المفاتيح في ذاكرة المخدم السيد للريديس، حتى لو انتهت منطقياً. ولن تعتبر موجودة، ولكن سوف يتم استعادة الذاكرة لاحقاً، وتكون قابلة للوصول بشكل متزايد وصريح. وبما أن هذه المفاتيح ليست أجزاء منطقية من مجموعة البيانات، يتم الإعلان عنها في خرج الأمر **INFO** وأمر **DBSIZE**.

عندما يقرأ المخدم التابع ملف **RDB** الذي ولّده السيد، لن يتم تحميل مجموعة المفاتيح. وكنتيجة لذلك من الشائع بالنسبة للمستخدمين الذين يملكون العديد من المفاتيح التي قد تنتهي صلاحيتها أن يروا القليل منها في المخدمات التابعة، ولكن لن يكون هنالك فرق منطقي فعلي في محتوى وحدات ريديس.

- التاريخ: 2017-04-18
- التصنيف: تكنولوجيا

#لغات البرمجة #علوم الحاسوب #الريديس #سلسلة الريديس

المصطلحات

- **القرص (Disk)** :(1)عبارة عن منطقة دائرية مسطحة من الغاز، والغبار و/أو النجوم. وقد يُشير هذا التعبير إلى المواد المحيطة بالنجم المتشكل حديثاً، أو المواد التي تتراكم بالقرب من ثقب أسود أو نجم نيوتروني، أو إلى المنطقة الكبيرة المحيطة بمجرة حلزونية والتي تحتوي أذرعاً حلزونية. (2)الشكل الظاهري الدائري للشمس، أو لكوكب، أو للقمر عندما يتم مشاهدتهم في السماء بوساطة تلسكوب.
- **الأيونات أو الشوارد (Ions)**: الأيون أو الشاردة هو عبارة عن ذرة تم تجريدها من الكترون أو أكثر، مما يُعطيها شحنة موجبة.وتسمى أيوناً موجباً، وقد تكون ذرة اكتسبت الكتروناً أو أكثر فتصبح ذات شحنة سالبة وتسمى أيوناً سالباً

المصادر

- redis.io
- الصورة
- الصورة

المساهمون

- ترجمة
  - ريم المير أبو عجيب
- مُراجعة
  - دانا أسعد
- تحرير
  - أنس عبود
- تصميم
  - محمد نور حماده
- نشر
  - مي الشاهد