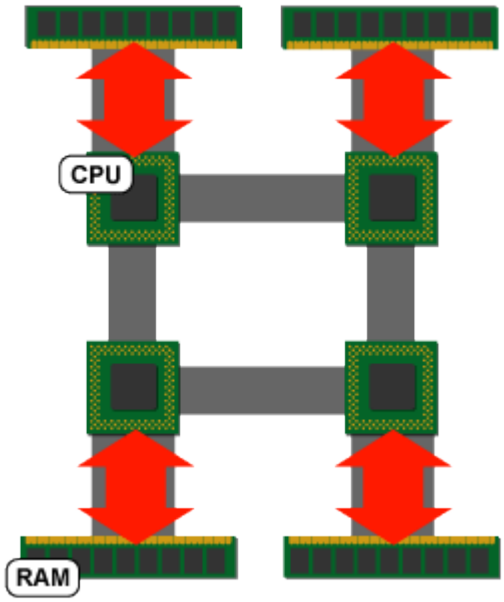
NUMA-эффект и анатомия дисковых бенчмарок

04 мая 2016



Что может быть проще измерения скорости записи файла? Засекаем время выполнения операции, не забывая о влиянии кэширования и о том, что основная часть работы может быть перенесена из функции записи в функцию закрытия файла, которую надо учесть в измерительном интервале. Все ли так очевидно? Результаты эксперимента, поставленного в тестовой лаборатории «Компостера» могут оказаться полезными как программистам, разрабатывающим собственные бенчмарки, так и системным администраторам, в обязанности которых входит выработка оптимальных стратегий использования серверов.

Суть эксперимента

Опыт выполнялся на двухпроцессорной NUMA-платформе с *64 GB*оперативной памяти. Установлены два 8-ядерных Intel Xeon Sandy Bridge с поддержкой Hyper-Threading, поэтому общее количество логических процессоров равно 2*8*2=32.

В силу особенностей лицензирования установленной ОС, системе доступно только *32GB* оперативной памяти, то есть «отрезана» память второго NUMA-узла и второй процессор использует неоптимальный доступ к памяти BSP-процессора. Возможно, это сыграло свою роль в том, что простейший пример уровня «Hello World» в области дискового бенчмаркинга, превратился в NUMA-оптимизированное приложение.

Напомним, мультипроцессорная система, в которой каждый из процессоров снабжен собственной подсистемой оперативной памяти, классифицируется как NUMA *(Non-Uniform Memory Access)* или система с неоднородным доступом к памяти. Какое это имеет отношение к производительности файловых операций? Как зависит скорость записи файла от того, какой из логических процессоров выполняет операцию?

Результаты все скажут сами.

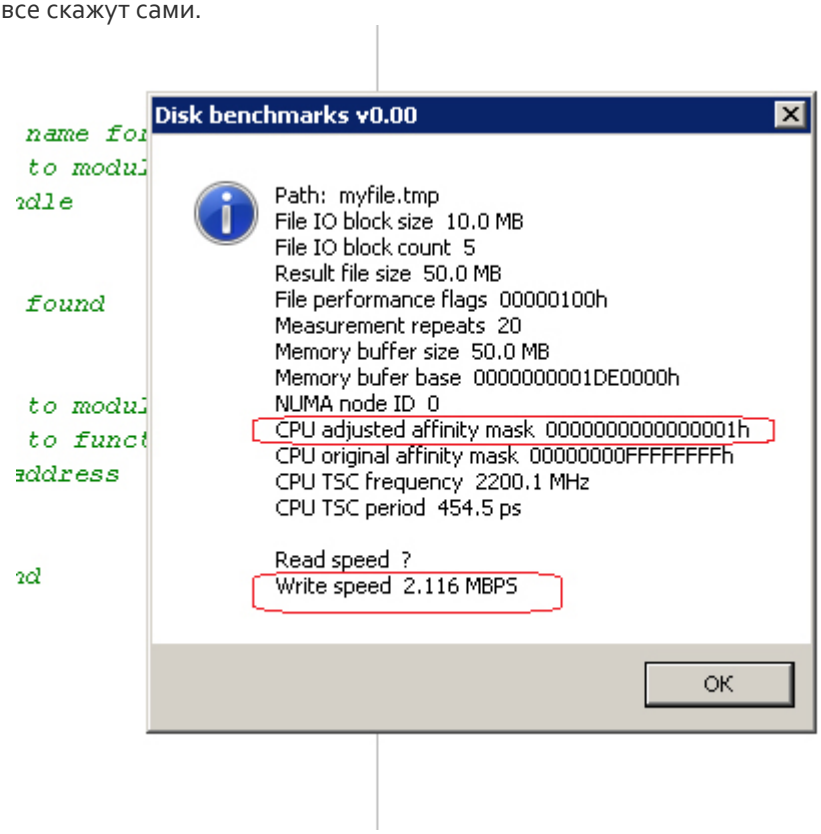


Рис.1 *Если параметр Affinity Mask задает процессор, относящийся к первому NUMA-узлу, скорость записи файла* ***2.116 мегабайт в секунду***

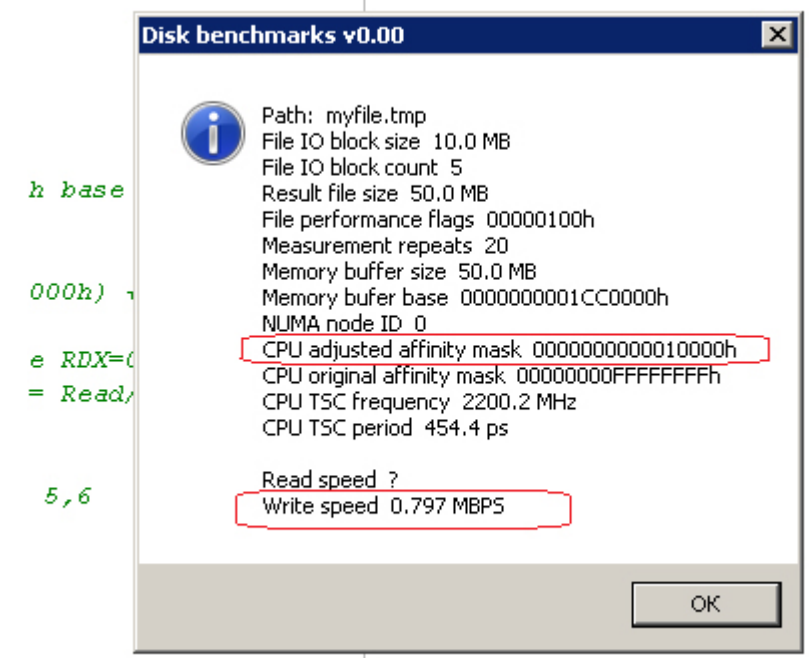


Рис.2 *Если параметр Affinity Mask задает процессор, относящийся ко второму NUMA-узлу, скорость записи того же файла* ***0.797 мегабайт в секунду***

Примечание. *В примере файл размером 50MB, записывается как 5 блоков по 10MB.*

О результате

В силу описанных особенностей стендовой платформы влияние NUMA-топологии оказалось особенно существенным. При размере файла 50 мегабайт операционная система имеет возможность задействовать кэширование. Исследование зависимости результата от размера файла и других параметров заслуживает стать темой отдельной публикации.

Абсолютные значения скоростей записи достаточно низкие, поскольку в данном опыте, в целях упрощения используется *синхронная последовательность запросов*, при которой следующий запрос на запись блока, не формируется до завершения обработки предыдущего запроса. Про этот фактор также необходимо помнить, как при написании собственных бенчмарок, так и при конфигурировании существующих.

Два фактора мультипроцессорной оптимизации

Формализуем акцент проведенных исследований в виде двух основных факторов.

1) Назначение логического процессора или группы логических процессоров, которые будут участвовать в выполнении исследуемой функции, в нашем случае это запись файла на диск. В описанном примере, ОС по умолчанию разрешает приложению использовать все доступные процессоры *(CPU original affinity mask = oooooooFFFFFFFFh)*, а наша программа намеренно ограничивает эти рамки, выбирая один процессор первого либо второго NUMA-узла, соответственно:

adjusted affinity mask = ooooooooooooooo1h либо ooooooooooo10000h.

Без явного контроля, повторяемость результатов бенчмарок ухудшается.

Напомним, *affinity mask* это битовый вектор, в котором каждому логическому процессору системы отведен один бит. Нулевое значение бита означает запрет участия данного процессора в выполнении программного модуля. Единичное значение разрешает использовать данный логический процессор. Маска для потока задается посредством WinAPI функции *SetThreadAffinityMask*. Также существует возможность задать маску для всех потоков процесса, функцией *SetProcessAffinityMask*.

2) Выбор NUMA-узла, в памяти которого будет размещен буфер, используемый для записи в файл. В рассматриваемом опыте влияние этого фактора минимально. Логично предположить, что имеет значение топологическая дистанция между выбранным NUMA-узлом и дисковым контроллером, который, напомним, способен взаимодействовать с оперативной памятью в режиме bus master. Для выделения блока памяти с указанием оптимального NUMA-узла, используется WinAPI функция *VirtualAllocExNuma*.

Исполнительным механизмом для целевой операции записи на диск в нашем примере являются три функции WinAPI: создание файла *CreateFile*, запись в файл *WriteFile*, закрытие описателя *CloseHandle*.

Файлы vs диски

Файловые операции ОС, сопровождающиеся распределением дискового пространства для записываемого файла, контролем и обновлением различных системных таблиц и т.п. занимают

достаточно большую долю времени, нивелируя зависимость бенчмарок от производительности накопителя и ухудшая повторяемость результатов тестов. Вспомним и о том, что дисковое пространство, отведенное файлу, может быть фрагментировано. Поэтому не удивительно, что результаты опытов могут быть весьма далеки от параметров, по которым принято различать ревизии спецификации SATA.

С другой стороны, для программиста или оператора бенчмарок, работа на уровне файлов значительно проще и позволяет абстрагироваться от реализации дисковой подсистемы, безболезненно переходя от локальных дисков к сетевым. В то время как получение истинной «аппаратной» скорости накопителя требует взаимодействия с ним на более низком уровне, как с устройством. Иначе, мы тестируем ОС API, а не диск. И чем производительнее накопитель, тем в большей степени выражен этот эффект.

Вместо послесловия

Измеренная скорость является функцией большого количества переменных, поэтому будем осторожны в трактовке результатов. Раздельная оценка факторов, связанных с кэшированием дисковых данных операционной системой и факторов, непосредственно относящихся к аппаратной реализации накопителей и контроллеров и их месту в NUMA-топологии платформы, станет предметом дальнейших исследований.

Даже такая банальность, как индикатор *HDD Led*, иногда может помочь оценить, какую долю рабочего времени приложение работает с накопителем и насколько кэширующие механизмы ОС нивелируют зависимость результатов теста от характеристик устройства, подменяя чтение-запись диска копированием блоков в оперативной памяти.

Выполняя операции с блоками, размер которых превышает размер доступной оперативной памяти, мы вынуждаем ОС интенсивно работать с накопителем.

С *теоретической* точки зрения нас интересует производительность дисков, а также *AHCI* или *NVME* хост-контроллеров. Чтобы минимизировать влияние операционной среды, тест потребуется провести в привилегированном режиме *Kernel Mode*, либо в среде *UEFI*, причем с доступом к диску на уровне секторов.

С *практической* стороны следует учитывать то, что обычный пользователь работает в среде ОС. Заметим, что в случае применения технологии Java, немаловажное значение приобретают особенности виртуальной машины JVM и, например, реализации библиотек ввода-вывода *java.io*, *java.nio*, известных практикующим программистам. Поэтому не менее полезно оценить *интегральную* производительность инфраструктуры, состоящей из накопителя, контроллера, ресурсов платформы, подсистемы *JRE (Java Runtime Environment)* и процедур ОС API. Показательно, что исследуемые эффекты, а следовательно и разрабатываемые методы оптимизации, применимы не только для дисковой подсистемы. Практически все сказанное, справедливо для высокопроизводительных решений в области локальных сетей и RDMA.