ABSTRACT

数据存储设备越来越智能，智能Flash存储设备（SmartSSD）将CPU处理器与DRAM存储打包到Smart SSD内部，使得用户程序可以在SmartSSD内部运行。本文致力于探索SmartSSD在关系型分析查询处理领域的机会与挑战。文章在三星Smart SSD上部署了基于微软SQL服务器的原型。实验结果表明通过将选择查询处理下推到Smart SSDs中可以获得显著的性能以及能耗收益。本文同时表明了SSD制造商可以采用哪些改变来增加使用SmartSSD用于数据处理应用的收益，同时给出了未来可能的研究机会。

Introduction

以往针对数据密集型应用优化的手段本质上还是将数据从存储端移动到Hos端（CPU）并且系统的硬件边界比较明显。随着现在永久存储、易失性存储、以及处理器之间的边界正在逐渐变得模糊，例如如今的手机在单个芯片上集成了诸多部件（Soc趋势）。本文聚焦处理器与非易失性存储在Smart SSD上的集成。SmartSSD是Flash存储设备，不同的是其在SSD中集成了存储与计算。本文探讨在SmartSSD中执行数据库操作的机会与挑战，机会主要为三方面：

- 与通用主机I/O接口（SAS，SATA）提供的带宽相比，SSDs可以提供更大的内部带宽。
- 将工作卸载到SmartSSD可能改变我们设计数据库服务器以及数据库应用的方式，SmartSSD的处理器相对低价，数据库服务器或应用使用SmartSSD性价比更高。
- 将数据处理下推到SmartSSD中可以减少整个数据库服务器/应用的能耗。

文章在SSD中实现的功能相对简单，将选择和聚合操作写入了SSD的硬件，扩展了SQL Server的执行框架以开发出可以端到端运行简单查询以及聚合操作的原型。对于此种查询，实验结果表明与没有smart功能的SSD相比使用SmartSSD可以获得2.7x的性能提升以及3.0x的能耗节省。文本也提出了在SmartSSD中运行数据处理程序的一些挑战：

- Smart SSD中可以使用的计算资源相对有限。
- 在SmartSSD中运行用户代码的硬件开发环境流程还不够成熟，可能是通用开发的一大挑战。
- 查询执行引擎以及查询优化器必须知道何时将操作下推到SSD。在SmartSSD中运行操作也需要扩展查询优化、DBMS缓冲区池缓存策略、事务处理，并且可能需要重新检查如何使用数据库压缩。

BACKGROUND： SSD ARCHITECTURE

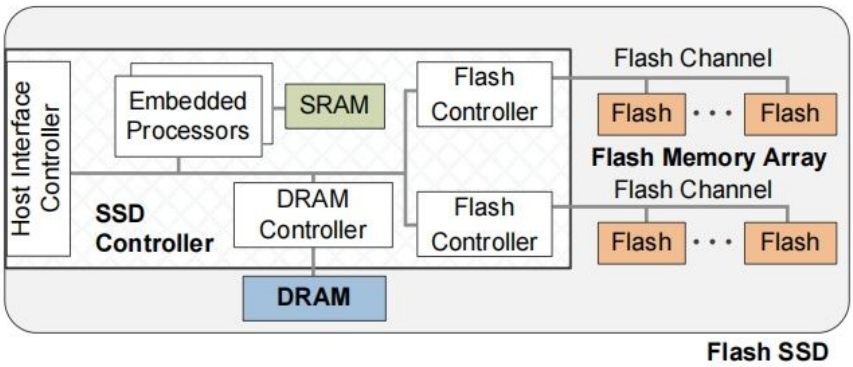

Figure 2: Internal architecture of a modern SSD

SSD内部结构大致如上，其中嵌入的处理器用来执行SSD固件代码，这些代码运行主机端接口协议、也运行Flash Translation Layer（FTL将主机端操作系统的逻辑块地址LBA映射到Flash内存中的物理块地址）。实时反应数据和程序代码被存储在SRAM中，处理器一般是一个32bit的RISC处理器例如多核ARM系列处理器。Flash Memory控制器负责flash内存与DRAM之间的数据移动，其核心功能包括运行纠删码ECC以及DMA。所有Flash通道共享访问DRAM所以来自Flash通道的数据移动会被序列化。NAND

Flash存储阵列是永久性存储媒介，每一个Flash chip都有多个block，每一个块都保存着多个page。擦除单元是block，硬件的读写操作粒度是page。

SMART SSDs FOR QUERY PROCESSING

SmartSSD运行时系统架构如图：

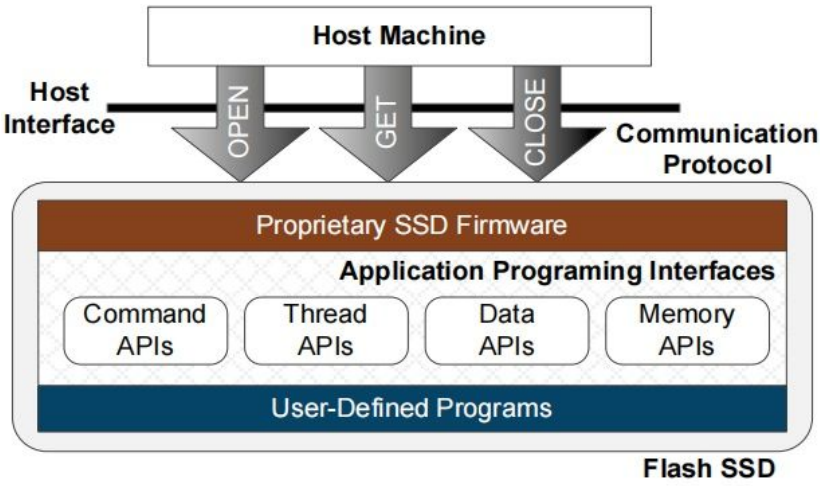

Figure 3: Smart SSD runtime framework

Communication Protocol

由于SmartSSD的关键概念是将常规SSD转换为组合计算和存储设备，我们需要一个标准的机制来支撑设备的运行时处理能力。我们已经开发了一个简单的基于会话的协议，它与标准的SATA/SAS接口兼容（但是可以扩展到PCIe）。该协议由三个命令组成——OPEN、GET、和CLOSE。

- **OPEN、CLOSE：**会话以OPEN命令开始，然后以CLOSE命令结束。一旦会话启动，就会授予包括运行用户定义程序所需的线程和内存（参见第3.2节中的线程和内存api）在内的运行时资源，然后将唯一的会话id返回给主机。CLOSE命令关闭与会话id关联的会话；它终止任何正在运行的程序，并释放该程序所使用的所有资源。会话关闭后，相应的会话id将无效，并且可以回收。
- **GET：**主机可以监视程序的状态，并检索程序通过GET命令生成的结果。该命令主要针对传统的块设备（基于SATA/SAS接口）设计，在这种情况下，存储设备是一个被动实体，只在主机发起请求时才进行响应。对于PCIe，可以引入一个更有效的命令（例如PULL）来直接利用设备启动的功能（例如，中断）。一个GET命令可以检索程序的运行状态和结果。使用不同的会话id，可以并行执行多个用户定义的程序。请注意，如果在智能SSD中没有可用的资源，这些程序可以被阻止。因此，轮询间隔应该是自适应的，这样它就不会引入很大的轮询开销或阻碍智能SSD操作的进程。在我们的实验中，轮询间隔被设置为10毫秒。

Application Programming Interface (API)

一旦命令通过智能SSD通信协议（第3.1节）成功传递到设备，SmartSSD运行时系统将以事件驱动的方式驱动用户定义的程序。用户程序可以使用SmartSSD API来进行命令管理、线程管理、内存管理和数据管理。API的设计理念是给程序更多的灵活性，以便最终用户程序更容易使用这些API。下面将简要描述这些API。

- **Command APIs**：每当智能SSD命令（即OPEN、GET和关闭）传递到设备时，智能SSD运行时系统将调用由用户定义程序注册的相应的回调函数。例如，打开和关闭命令分别触发用户定义的打开和关闭函数。相反，GET命令调用函数来填充程序的运行状态，并在可用时将结果传输到主机。
- **Thread APIs**：一旦打开会话，智能SSD运行时系统将创建一组工作线程和一个专用于会话的主线程。由运行时系统管理的所有线程都是非抢占式的。当SmartSSD命令到达时（请参见上面的命令API），或者当数据页从flash加载到DRAM时（请参见下面的数据API）时，工作线程会被调度。

- **Memory APIs**：智能SSD设备通常有两种类型的内存模块，一个小的快速SRAM（例如，ARM的紧密耦合内存），和一个大的慢速DRAM。在一个典型的场景中，DRAM主要用于存储数据页面，而SRAM则用于频繁访问的元数据，如数据库表模式。一旦会话被打开，预定义的内存量将分配给会话，当会话关闭时，该内存将返回到智能SSD运行时系统（不允许使用malloc和空闲分配动态内存）。
- **Data APIs**：多个数据页面可以从flash并行加载到DRAM。在这里，并行性的程度取决于在智能SSD中使用的闪存通道的数量。加载后，页面将被固定，以确保它们不会退出DRAM。处理页面后，必须取消它，才能释放将页面放回设备所需的内存。否则，智能SSD操作可能会被阻止，直到有足够的内存可用于后续操作。

EVALUATION

在我们的实验中，我们使用了TPC-H基准[30]中定义的线性数组表和三个合成表（合成4、合成16和合成64），它们分别由4个整数列、16个整数列和64个整数列组成。此外，我们还创建了三个合成表，分别称为合成4、合成16和合成64，每个表都有400M的元组。这些表的大小分别为10GB、30GB和110GB、合成16和合成64表。

Table 1: Maximum sequential read bandwidth with 32-page (256KB) I/Os.

	SAS HDD	*SAS SSD*	*(Internal) Smart SSD*
Seq. Read (MB/sec)	80	550	1,560

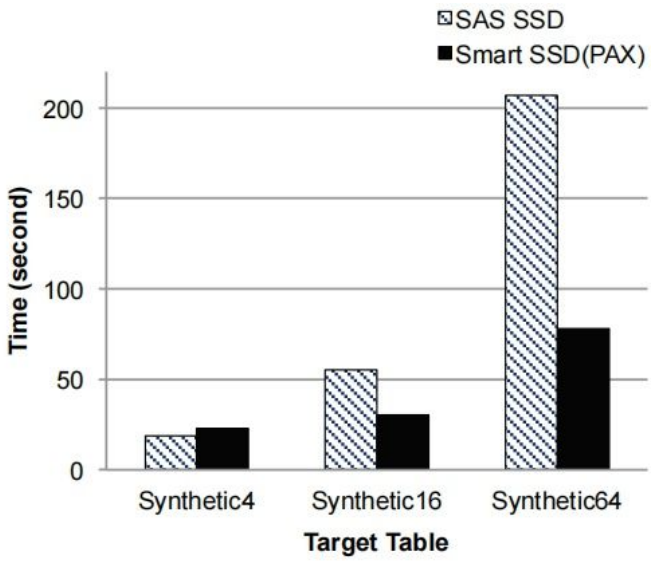

Figure 4: End-to-end elapsed time for a selection query at a selectivity of 0.1% with the three synthetic tables Synthetic4, Synthetic16, and Synthetic64.

从表1中可以看出，Smart SSD的内部顺序读取带宽分别比HDD和SSD快19.5X和2.8倍。此值可用作此智能SSD可能提供的性能提升的上限。如图1所示，随着时间的推移，SSD和智能SSD之间的差距很可能会增长到比2.8X大得多的数字。我们还注意到，这里的改进（约为2.8X）远小于图1中所示的间隙（大约为10X）。造成这种差距的原因是，对DRAM的访问由所有的闪存通道共享，目前在这个SSD设备中，一次只能有一个通道被激活（回顾第2节中的讨论），这成为了瓶颈。人们可以通过增加到DRAM的带宽或添加更多的DRAM总线来解决这个瓶颈。

Selection Query

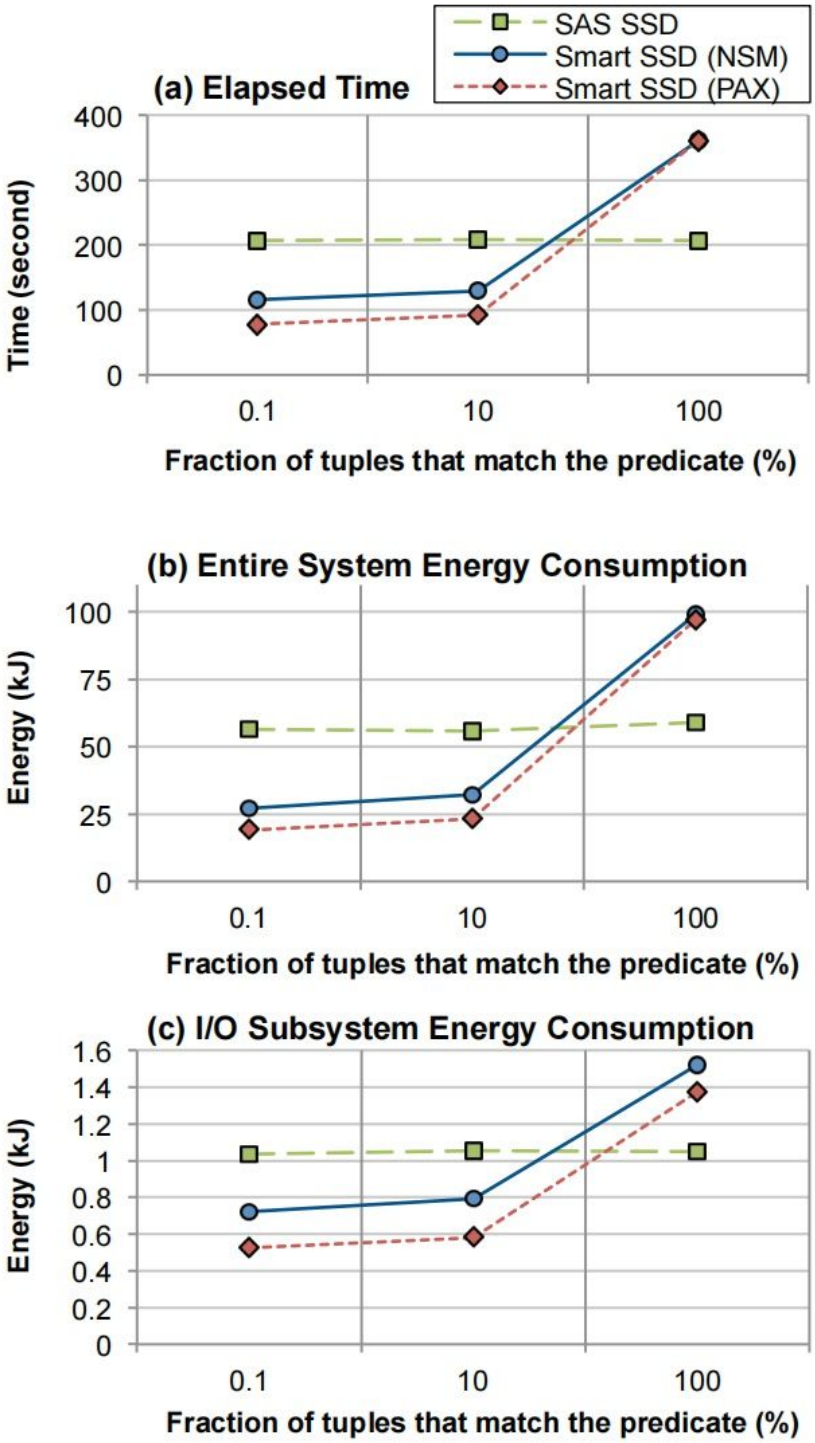

Figure 5: End-to-end (a) query execution time, (b) entire system energy consumption, and (c) I/O subsystem energy consumption for a selection query on the Synthetic64 table at various selectivity factors.

- **Effect of Tuple Size:**智能SSD的性能改进来自于更快的内部I/O，而智能SSD中ARM核心的低计算能力饱和了其性能。与普通的SSD案例相比，智能SSD在发送到主机之前，必须计算从闪存芯片获取的页面中的数据。在合成4表中，每个数据页面上有323个元组，而基于智能ssd的执行策略现在必须在每个页面上花费更多的处理周期，这使CPU饱和。现在，智能SSD中的查询（在合成4表上）在CPU资源上受到了瓶颈。
- **Effect of Varying the Selectivity Factor:**从图5 (a)中的一个有趣的观察结果是，对于智能SSD案例，使用PAX布局提供了比NSM布局更好的性能，高达32%。由于在PAX布局的情况下，页面中列

的所有值都是连续存储，我们能够使用LDM指令一次加载多个值，从而减少了（慢）DRAM访问的次数。考虑到SSD中的高DRAM延迟，柱状PAX布局比基于行的布局更有效。

Selection with Aggregation Query

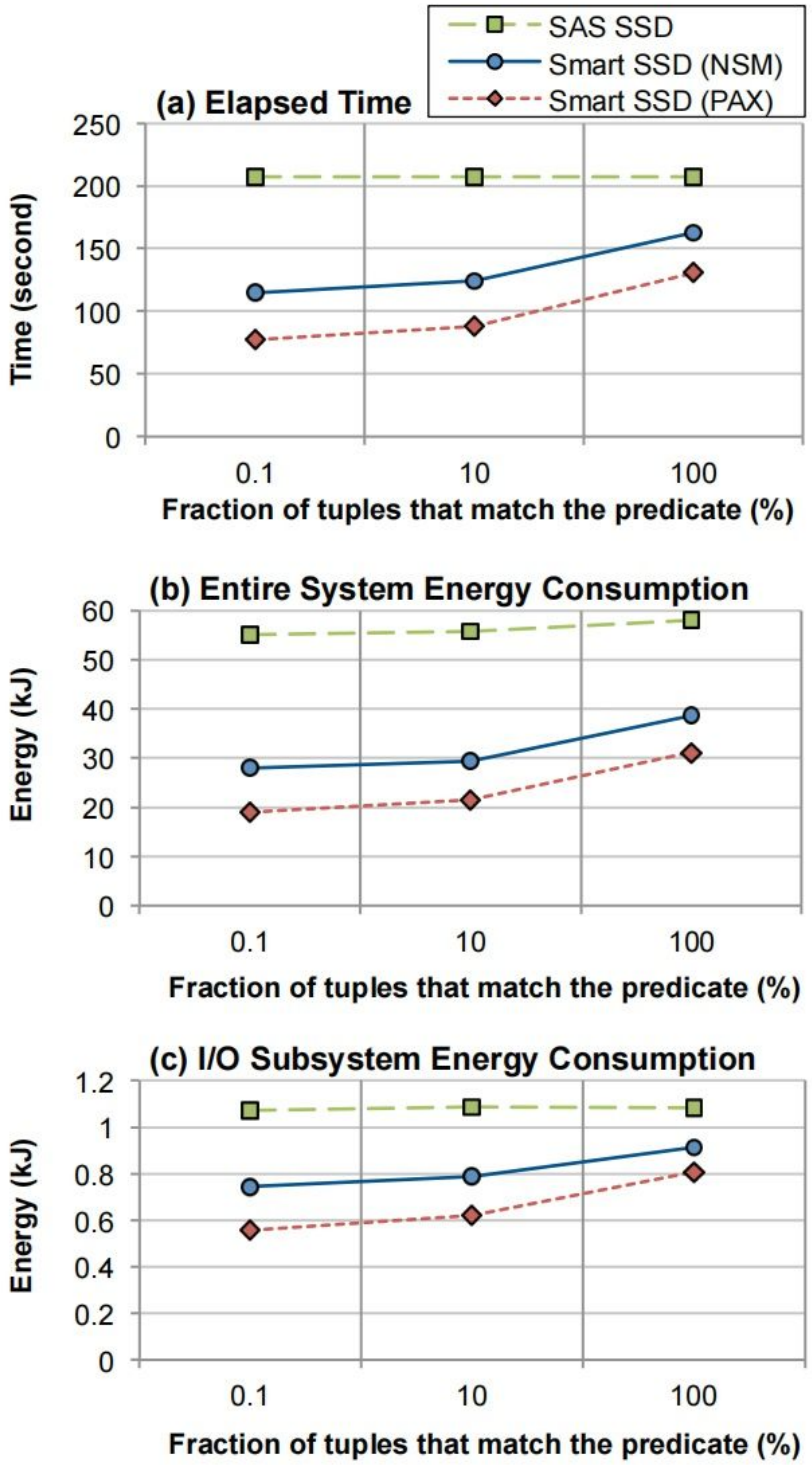

Figure 6: End-to-end (a) query execution time, (b) entire system energy consumption, and (c) I/O subsystem energy consumption for a selection with aggregate query on the Synthetic64 table at various selectivity factors.

简单选择之间巨大的性能差异表明，对于聚合查询，智能SSD比硬盘和SSD情况下即使在100%的选择性下也有更好的性能。原因是，**聚合查询的输出远小于选择查询的输出。**因此，与将数据从智能SSD传输到主机相关联的选择查询的I/O成本要高得多，这降低了智能SSD的好处。**此外，数据页面中的元组数量对使用智能SSD实现的性能改进有很大的影响。**

DISCUSS CONCLUSION AND FUTURE WORK

一个重要的观察是，SmartSSD内的处理能力很快成为性能瓶颈，特别是当选择谓词匹配许多输入元组或每页有大量的处理数据时（例如，合成4表）。没有L1/L2缓存的低性能嵌入式处理器和访问DRAM内存的高延迟成本很快成为了瓶颈。此外，正如在第4.2.1节中所讨论的，**当查询需要的每个数据页面的计算更少时，智能SSD可以获得更大的好处。**在智能SSD中运行代码所需的开发环境需要进一步的开发。

在DBMS方面，如果缓冲区池中的数据副本比SSD中的数据更最新，那么将查询处理推送到SDD可能是不可行的。类似地，如果没有与DBMS事务管理器进行适当的协调，那么在SSD中就无法处理具有任何更新的查询。如果数据库是不可变的，那么其中一些这些问题将变得更容易处理。如果全部或部分数据已经缓存在缓冲池中，那么将处理推送到智能SSD可能没有好处（从性能和能源消耗的角度来看）。主机可以简单地是跨SmartSSD组进行计算的协调器，使系统看起来像一个并行DBMS，主节点是主机服务器，并行系统中的工作节点是智能ssd。SmartSSD可以在内部运行轻量级隔离SQL引擎，这些轻量级隔离引擎由主机节点进行全局协调。

未来也可以探索以下几个方向：

- 扩展查询优化器以将操作推送到智能SSD
- 为在智能SSD内工作的各种操作设计相应的算法
- 考虑并发查询的影响
- 检查在智能SSD内运行的操作对缓冲区池管理的影响
- 考虑到各种存储布局的影响