TDengine 白皮书

1 大数据时代的挑战

随着移动互联网的普及，数据通讯成本的急剧下降，以及各种低成本的传感技术和智能设备的出现，除传统的手机、计算机在实时采集数据之外，从手环、共享自行车、出租车、智能电表、环境监测设备到电梯、大型设备、工业生产线等都在源源不断的产生海量的实时数据发往云端。这些海量数据是企业宝贵的财富，能够帮助企业实时监控业务或设备的运行情况，生成各种维度的报表，而且通过大数据分析和机器学习，对业务进行预测和预警，帮助企业进行科学决策、节约成本并创造新的价值。

仔细研究发现，所有机器、设备、传感器、以及交易系统所产生的数据都是时序的，而且很多还带有有位置信息。这些数据具有明显的特征，1: 数据是时序的，一定带有时间戳；2：数据是结构化的；3: 数据很少更新删除；4：写多读少；5：用户关注的是一段时间的趋势，而不是某一特点时间点的值；6: 数据是有保留期限的；7：数据的查询分析一定是基于时间段和地理区域的；8：除存储查询外，还往往需要各种统计和实时计算操作；9：数据量巨大，一天采集的数据就可以超过 100 亿条。

看似简单的事情，但由于数据记录条数巨大，导致数据的实时写入成为瓶颈，查询分析极为缓慢，成为新的技术挑战。传统的关系型数据库或 NoSQL 数据库以及流式计算引擎由于没有充分利用这些数据的特点，性能提升极为有限，只能依靠集群技术，投入更多的计算资源和存储资源来处理，企业运营维护成本急剧上升。

2 TDengine 特点

TDengine 正是涛思数据面对这一高速增长的时序数据市场和技术挑战推出的创新性的大数据处理产品，它不依赖任何第三方软件，也不是优化或包装了一个开源的数据库或流式计算产品，而是在吸取众多传统关系型数据库、NoSQL 数据库、流式计算引擎、消息队列等软件的优点之后自主开发的产品，在时序空间数据处理上，有着自己独到的优势。

- TDengine 定义了创新的时序数据存储结构，通过采用无锁设计和多核技术，让数据插入和查询的速度比现有专业的时序数据库提高了十倍以上；
- TDengine 可将多个时序数据流进行实时聚合计算，提供强大的流式计算功能。

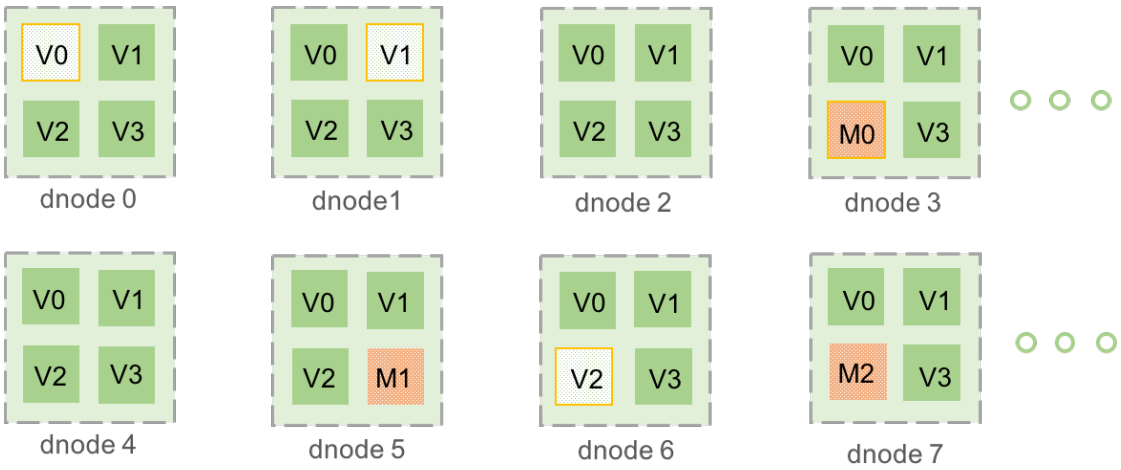
- 将大数据处理所需要的数据库、消息队列、缓存、流式计算等功能融合一起，应用无需再集成这些功能的软件，大幅降低应用开发难度；
- 基于分布式集群的设计，TDengine 不仅保证了系统处理能力的水平扩展，而且让数据库不再依赖昂贵的硬件和存储设备，不存在任何单点瓶颈和故障；
- 通过实时数据与标签分离，列式存储和先进的压缩算法，TDengine 存储空间不到通用数据库的 10%；
- 追求极致的用户体验，让运营维护智能化，扩容、升级、数据同步、恢复、迁移等都轻松搞定。

采用 TDengine，可将典型的物联网、车联网、工业互联网大数据平台的整体成本降至现有的 1/5。同样的硬件资源，TDengine 能将系统处理能力和容量增加五倍以上。

为缩短学习曲线，TDengine 采用的是传统数据库的数据模型和 SQL 语法，支持 ODBC 和 JDBC，而且 Driver 提供的 API 与 MySQL 完全一样。为支持各种大数据分析软件，并与其他数据库互联，TDengine 提供 Hadoop 和 Spark 连接器，TDengine 成为他们的数据源。

3 TDengine 系统结构

TDengine 是基于硬件、软件系统不可靠、一定会有故障的假设进行设计的，是基于任何单台计算机都无足够能力处理海量数据的假设进行设计的，因此 TDengine 从研发的第一天起，就是按照分布式高可靠架构进行设计的，是完全去中心化的。TDengine 整个系统结构如图所示，下面对一些基本概念进行介绍。



物理节点：集群里的任何一台物理机器(dnode)，根据其具体的 CPU、Memory、存储和其他物理资源，TDengine 将自动配置多个虚拟节点。

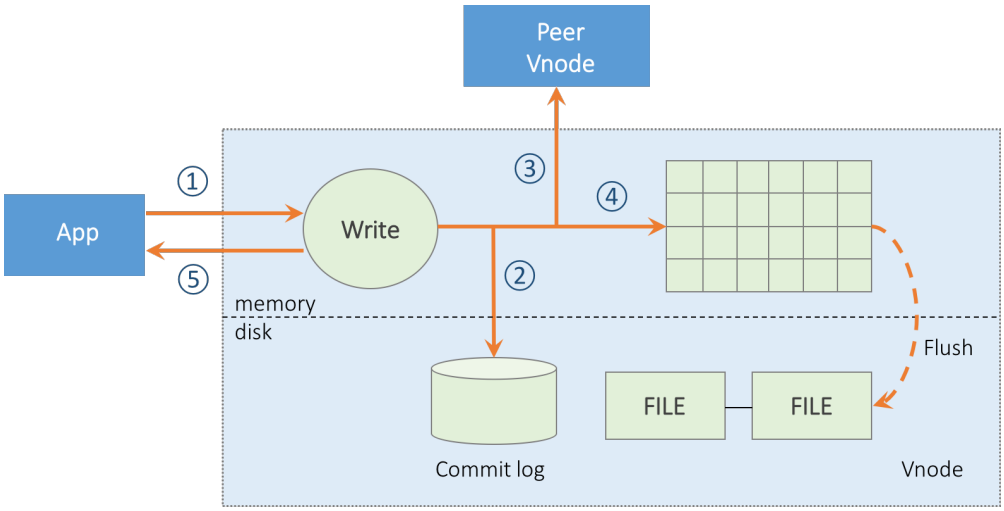
虚拟数据节点：存储具体的时序数据，所有针对时序数据的插入和查询操作，都在虚拟数据节点上进行（图例中用 V 标明）。位于不同物理机器上的虚拟数据节点可以组成一个虚拟数据节点组（如图例中 dnode0 中的 V0, dnode1 中的 V1, dnode6 中的 V2 组成了一个组），虚拟节点组里的虚拟节点的数据是以 P2P 的方式完全实时同步的，以保证一份数据在多台物理机器上有拷贝，而且即使一台物理机器宕机，总有位于其他物理机器上的虚拟节点能处理数据请求，从而保证系统运行的高可靠性。

虚拟管理节点：负责所有节点运行状态的采集、节点的负载均衡，以及所有 Meta Data 的管理，包括用户、数据库、表的管理（图例中用 M 标明）。当应用需要插入或查询一张表时，如果不知道这张表位于哪个数据节点，应用会连接管理节点来获取该信息。Meta Data 的管理也需要有高可靠的保证，系统采用 Master-Slave 的机制，容许多到 5 个虚拟管理节点组成一个虚拟管理节点集群（如图例中的 M0, M1, M2)。这个虚拟管理节点集群的创建是完全自动的，无需任何人工干预，应用也无需知道虚拟管理节点具体在哪台物理机器上运行。

集群对外服务 IP：整个系统可以由多台甚至数万台服务器组成，但对于应用而言，只需要提供整个集群中任何一台或两台服务器的 IP 地址即可。 TDengine 将根据应用的请求，自动的将请求转发到相应的一个甚至多个节点进行处理，包括聚合、计算操作等。这些复杂的分发和路由对应用是完全透明的。

4 TDengine 存储结构

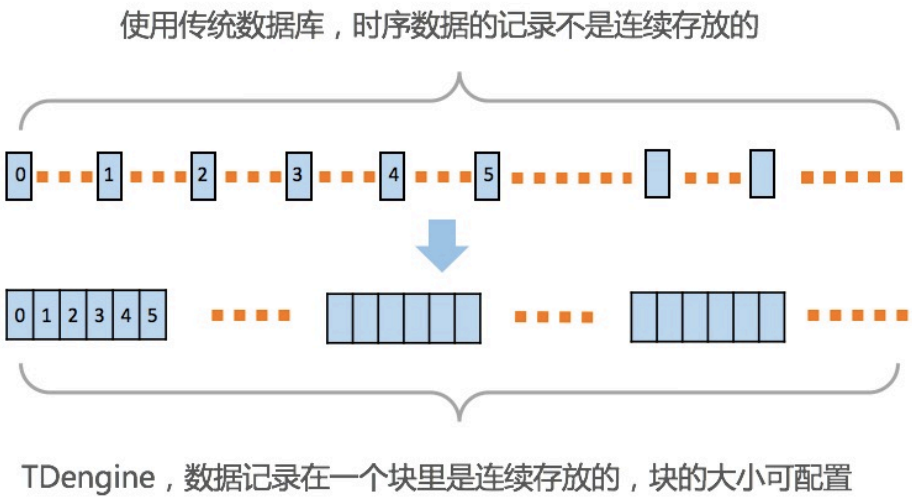
为提高压缩和查询效率，TDengine 采用列式存储。但与众多的时序数据库不一样的是，TDengine 将每一个采集点的数据作为数据库中的一张独立的表来存储，对于一个采集点而言，无论在内存还是硬盘上，数据点在介质上是连续存放的，这样大幅减少随机读取操作，数量级的提升读取和查询效率。整个写的过程如图所示：



写入数据时，先将数据点写进 Commit 日志，然后转发给同一虚拟节点组里的其他节点，再按列写入分配的内存块。当内存块的剩余空间达到一定临界值或设定的 commit 时间时，内存块的数据将写入硬盘。内存块是固定大小(如 16K)的，但依据系统内存的大小，每个采集点可以分配一个到多个内存块，采取 LRU 策略进行管理。在一个内存块里，数据是连续存放的，但块与块是不连续的，因此 TDengine 为每一个表在内存里建立有块的索引，以方便写入和查询。

数据写入硬盘是以添加日志的方式进行的，以大幅提高落盘的速度。为避免合并操作，每个采集点（表）的数据也是按块存储，在一个块内，数据点是按列连续存放的，但块与块之间可以不是连续的。 TDengine 对每张表会维护一索引，保存每个数据块在文件中的偏移量，起始时间、数据点数、压缩算法等信息。每个数据文件仅仅保存固定一段时间的数据(比如一周，可以配置)，因此一个表的数据会分布在多个数据文件中。查询时，根据给定的时间段，TDengine 将计算出查找的数据会在哪个数据文件，然后读取。这样大幅减少了硬盘操作次数。多个数据文件的设计还有利于数据同步、数据恢复、数据自动删除操作，更有利于数据按照新旧程度在不同物理介质上存储，比如最新的数据存放在 SSD 盘上，最老的数据存放在大容量但慢速的硬盘上。通过这样的设计，TDengine 将硬盘的随机读取几乎降为零，从而大幅提升写入和查询效率，让 TDengine 在很廉价的存储设备上也有超强的性能。

为减少文件个数，一个虚拟节点内的所有表在同一时间段的数据都是存储在同一个数据文件里，而不是一张表一个数据文件。但是对于一个数据节点，每个虚拟节点都会有自己独立的数据文件。



5 数据分区、水平扩展

为处理每日高达数亿条的海量数据，数据必须在多个节点存放。在 TDengine 里，数据是按照每个采集点（表）来存放的。一张表（一个采集点）的数据，即使每秒产生一百个字节的数据量，一年也才 3G 的数据量，压缩后，往往还不到 300M，因此 TDengine 里，一个表是不跨节点存储的，以便于单张表的快速高效的插入、查询和计算。

为更好的数据分区，TDengine 采用了虚拟数据节点的设计。一个虚拟数据节点包含多个表，表的数量可以配置。根据其计算和存储资源，一个物理节点将被划分为多个虚拟数据节点。虚拟数据节点的设计带来几大优势，1）更好的支持硬件异构环境，资源多的服务器可以创建更多的虚拟节点；2）恢复一个宕机的节点，可以让众多的其他节点参与进来，大大加快速度；3）如果撤掉一个数据节点，该节点上的虚拟节点将被相当均匀的迁移到其他节点上去；4）新增一个数据节点，负载过热的节点的上的部分虚拟节点将被整体迁移过来。这一切让负载更加均衡，让数据同步变得更加高效。

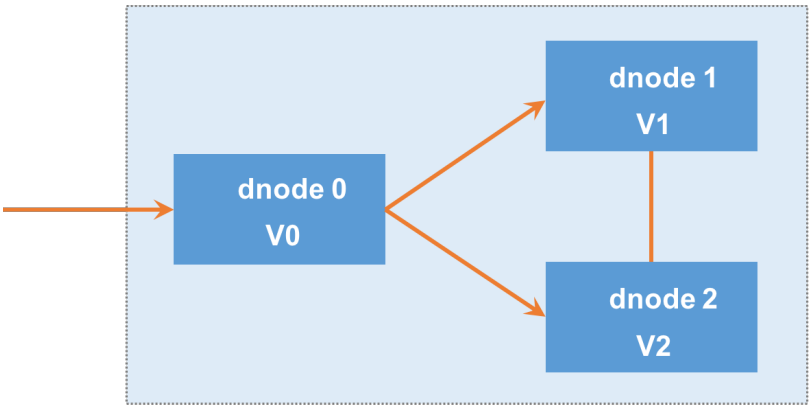
与传统的数据库相似，用户可以可以创建多个数据库，每个库里面，可以创建多个表。一个库可以横跨多个虚拟数据节点，但一个虚拟数据节点仅仅属于一个数据库。当用户添加一个表时，管理节点将查看已经分配的虚拟节点里是否还有空位，如果有，就将该表分配到这虚拟节点。如果这个库的所有虚拟节点都没有空位，管理节点将根据负载均衡的策略(随机、轮询等)来分配一个新的虚拟节点给该库，然后将该表分配到新的虚拟节点里。由于一台物理主机有多个虚拟数据节点，这种策略能保证负载均匀分布。

管理节点负责整个系统的负载均衡，虚拟数据节点的增加、删除、迁移、合并与拆分。管理节点并不保存每个采集点采集的数据，只是管理虚拟节点，而且即使宕机，也不会影响现有各虚拟节点的数据插入和查询操作。各个采集点或应用从管理节点获取分配的虚拟数据节点信息后，然后直接与虚拟数据节点通讯，直接将数据插入数据库，对于查询操作也是如此。因此，系统容量以及吞吐率与虚拟数据节点的个数成正比，整个系统是水平扩展的

6 高可靠系统

为保证数据节点的高可靠性，TDengine 引入了虚拟数据节点组的概念，并采用 P2P 的方式进行数据同步。一个虚拟节点组由处于不同物理主机上的虚拟数据节点组成，虚拟数据节点个数就是数据冗余的个数(Replication Factor，一般大于 2)。在一个虚拟节点组里，各个虚拟数据节点通过心跳包实时知道对方的状态，而且没有主次之分，每个虚拟数据节点都可以接受数据写入和查询的请求，如果一个虚拟数据节点收到数据写入的请求，该请求会被立即转发给其他虚拟数据节点，然后在本地存储处理。当应用连接 TDengine 系统时，对于要操作的任何一张表，系统会给应用提供该表所属的虚拟数据节点组里各个虚拟节点的 IP 地址（如果 replication factor 为 3，就会有 3 个 IP 地址），如果链接其中一个失败或者操作失败，应用会尝试第二个、第三个，只有所有节点失败才会返回失败。这样保证虚拟数据节点组里任何一台机器宕机，都不会影响对外的服务。这些复杂的重新连接流程都被 TDengine Driver 包装隐藏起来，应用开发者无需写程序来实现。

TDengine 采取 Master-Slave 方式实现多个副本之间的实时数据同步，为保证效率，采取的是最终一致性，而不是强一致。当一台主机重启时，每个虚拟数据节点都会检查自己数据的版本是否与其他虚拟节点一致，如果版本不一致，需要同步后才能进入对外服务状态。在运行过程中，由于各种原因，数据仍然可以失去同步，这种不同步会在收到转发的写入请求时被发现，一旦被发现，版本低的虚拟数据节点将马上停止对外服务，进入同步流程，同步完后，才会重新恢复对外服务。同步过程中，高版本的节点还可以正常的对外提供服务。



接收的数据会被转发到虚拟节点组里的其他节点

管理节点负责存储 Meta 数据，同时根据每个数据节点状态来负责负载均衡，因此也要保证其高可靠性。多个虚拟管理节点组成一个虚拟管理节点组，因为 Meta 数据可以被多个应用同时更新，因此 TDengine 采用的是 Master-Slave 模式实现虚拟管理节点的数据同步。写的操作，只有 Slave 节点写入成功后，Master 节点才会返回成功，从而保证数据的强一致性。如果 Master 节点宕机，系统有机制保证其中一个 Slave 会立即被选举为 Master，从而保证系统写操作的高可靠性。

由于 Meta 数据量并不大，Meta 数据虽然需持久化存储，但将其完全保存在内存，以保证查询操作的高效。在应用侧，为避免每次数据操作都访问管理节点，TDengine Driver 将必要的 Meta 数据都会缓存在本地，只有当需要的 Meta 数据不存在或失效的情况下，才会访问管理节点，这样大大提高系统性能。

管理节点在集群中存在，但对于应用和系统管理员而言，是完全透明的。整个系统会自动在物理节点上创建虚拟管理节点以及虚拟管理节点组。

7 Metric：多表聚合

各个数据采集点的时钟是很难同步的，为保证其时序，而且为保证单一采集点的数据在存储介质上的连续性，TDengine 要求每个数据采集点单独建表，这样能极大提高数据的插入速度以及查询速度，但是导致系统表的数量猛增，让应用对表的维护以及聚合、统计操作难度加大。为降低应用的开发难度，TDengine 引入了 Metric 的概念。

Metric 是表的集合，包含多张表，而且这集合里每张表的 schema 是一样的。同一类型的采集设备可创建一个 Metric。与表一样，包含 Schema，但还包含标签信息。Schema 定义了表的每列数据的属性，如温度、压力等，而标签信息是静态的，属于 Meta Data，如采集设备的型号、位置等。TDengine 扩展了标准 SQL 的 table 的定义，创建时，除指定 Schema 外，还可以带关键词 tags 来指定有哪些标签。如：

```
create table m1(ts timestamp, pressure int, rpm int) tags (model binary(8), color binary(8))
```

上述 SQL 创建了一个 Metric m1，带有标签 model 和标签 color。为某一个具体的采集点创建表时，可以指定其所属的 Metric 以及标签的值，比如：

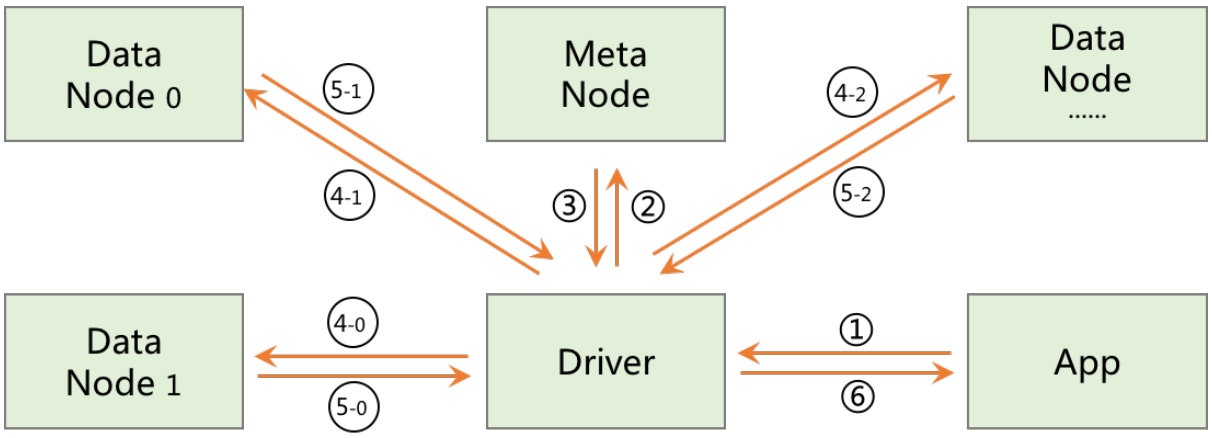
```
create table t1 using m1 tags ('apple', 'red')
```

上述 SQL 以 Metric m1 为模板，创建了一张表 t1，这张表的 Schema 就是 m1 的 Schema，但标签 id 设为 apple，标签 color 设为 red。插入数据时，仍然按照正常的方式进行插入。但查询时，除传统的表的查询外，还可以基于标签对 Metric 进行各种聚合查询或统计。如：

```
select avg(pressue) from m1 where model='apple' interval(5m) group by color
```

上面这个 SQL 语句表示将标签 model 值为 apple 的所有采集点的记录的每 5 分钟的平均值计算出来,并按照标签 color 进行分组。

对于 Metric 的查询操作，完全与正常的表一样。但一个定义的 Metric 可以包含多张表（多个数据采集点），应用可以通过指定标签的过滤条件，对一个 Metric 下的全部或部分表进行聚合或统计操作，这样大大简化应用的开发。其具体流程如下图所示：



1：应用将一个查询条件发往系统；2: Driver 将查询的过滤条件发往 Meta Node（管理节点)；3：管理节点将符合查询过滤条件的表的列表发回 Driver(包含每个表对应的数据节点的 IP 地址)；4：这些返回的表可能分布在多个数据节点，Driver 将计算的请求发往相应的多个数据节点；5：每个数据节点完成相应的聚合计算，将结果返回给 Driver；6：Driver 将多个数据节点返回的结果做最后的聚合，将其返回给应用。

8 数据模型

TDengine 采用的仍然是传统的关系型数据库的模型。用户需要根据应用场景，创建一到多个库，然后在每个库里创建多张表，创建表时需要定义 Schema。对于同一类型的采集点，为便于聚合统计操作，可以先定义 Metric，然后再定义表。

不同的采集点往往具有不同的数据特征，比如有的采集点数据采集频率高，有的数据保留时长较长，有的采集数据需要 3 份备份，而有的数据一份备份即可，有的采集点一条记录很大，而有的采集点的记录仅仅 16 个字节，很小。为让各种场景下 TDengine 都能最大效率的工作，TDengine 建议将不同数据特征的表创建在不同的库里。创建一个库时，除 SQL 标准的选项外，应用还可以指定保留时长、数据备份的份数、cache 大小、是否压缩等多种参数。

TDengine 对库的数量、Metric 的数量以及表的数量没有做任何限制，而且其多少不会对性能产生影响，应用按照自己的场景创建即可。

9 实时流式计算

在存储的原始数据上，TDengine 可以做各种计算，目前支持的主要操作包括：

- Avg：以每个采样时间范围内的 value 的平均值作为结果
- Dev：以每个采样时间范围内的 value 的标准差作为结果
- Count：以每个采样时间范围内的点的数目作为结果
- First：以每个采样时间范围内的第一个 value 作为结果
- Last：以每个采样时间范围内的最后一个 value 作为结果
- LeastSquares：对每个采样时间范围内的 value 进行最小二乘法的拟合
- Max：以每个采样时间范围内的 value 的最大值作为结果

- Min：以每个采样时间范围内的 value 的最小值作为结果
- Percentile：每个采样时间范围内的 value 的第 p 百分位数作为结果。
- Sum：以每个采样时间范围内的 value 的总和作为结果
- Diff：以每两个相邻的 value 的差值作为结果
- Div：以每个 value 除以一个除数作为结果
- Scale：以每个 value 乘以一个倍数作为结果
- 基于多个采集点数据的四则运算表达式

TDengine 还可对一个或多个数据流进行实时聚合、统计等计算，并将计算出的衍生数据当做新的数据保存进 TDengine，以便后续的操作。实时计算与聚合查询很类似，只是后台定时进行，并自动滑动计算窗口的起始点。工作方式与其他流式计算引擎的 Sliding Window 相似。

实时计算可以通过一个简单的创建表的操作来实现。如：

```
create table d1 as select avg (pressure) from t1 interval (60s) sliding(10s)
```

上述 SQL 表示将表 t1 里字段 pressure 每 10 秒钟(每次滑动的时间间隔）将过去的 60 秒钟（聚合计算的时间间隔）的数据平均值计算出来并写入表 d1。计算出的衍生数据可以与其他原始数据或计算出的衍生数据进行再次计算。

10 便捷的安装、部署、维护

TDengine 是在 Linux 上开发的，任何 Linux 系统都可以运行，而且不依赖任何第三方软件，也不是在某个开源项目上包装出来的产品。获得安装包并解压后，只需执行 install 脚本就一切搞定，极其简单。

安装后，会在安装的机器上自动创建虚拟数据节点和管理节点，开发者就可以使用了，能满足一般性的需求。但如果数据量大，就需要将软件安装到多台主机。这时也只需要在每台机器配置好 Master IP，系统管理员打开 TDengine Shell，将新添加的主机添加进系统即可。如果要撤销一个物理节点，登录 TDengine Shell，将其删除即可，极其简单。传统数据库所需要的数据分区、数据迁移等等都一概不存在了。

因为数据是自动同步到多个节点的，系统管理员不用担心数据的丢失，也不用制定备份和数据恢复策略，一切全自动进行。

如果软件需要升级，只要在 TDengine Shell 里将新版本上传即可。管理节点将挨个把每个节点的软件进行升级，而且整个系统的服务将不停止，服务不受任何影响。如果要更换设备，只需将其拔除，安装上软件后，将新设备重新插入即可。换言之，TDengine 完全支持在线升级以及硬件的热插拔，从而保证服务的 7*24 的不间断运行。

开发人员需要做的是定义表的结构，根据具体场景，配置好各种参数，让系统性能达到最优。系统管理员只需要关注与硬件相关的报警信息，对于经常出问题的服务器或硬盘，进行更换而已。使用 TDengine，整个系统的运维工作变得极为简单，将大大降低运营成本。

11 更多亮点

订阅模式：与标准的数据库不同，TDengine 还提供一种订阅模式。应用程序可以订阅数据库某张表的内容，一旦该表有新的记录，应用将立即得到通知。同一个表可以被多个应用订阅。与流行的消息中间件 Kafka 一样，订阅采取的是 pull 而不是 push 模式。Kafka 的 publish 操作由数据库插入操作代替。由于 TDengine 具有极高的插入速度，高于 Redis 和 Kafka，因此对于时序数据的应用，可以不再需要消息缓存的中间件。通过采用订阅模式，TDengine 本身也可以作为一个消息队列中间件来使用。

异步插入：为避免网络延迟带来的性能下降，更好的提高数据插入速度，TDengine 还提供一组 API 让应用异步插入数据。当应用调用插入的 API 时，将立即返回，等记录成功插入后，TDengine 将调用应用提供的回调函数通知应用。采用异步 API，可以把性能大幅提高。

Nagle 算法：时序数据应用场景里，每条记录一般都很小，很多不到 20 字节，因此整个系统处理的是大量的小数据包。为了更进一步提高性能，减少网络 IO 次数，TDengine 采用了类似 TCP 协议的 Naggle 算法，客户端将缓存插入请求，只有记录

的大小超过一定的大小或者缓存时间超过 100 毫秒，被缓存的插入请求才会被发往系统。对于时间要求很高的应用，该功能可以关闭。

12 参数指标

- 支持数据类型：tinyint, smallint, int, bigint, float, double, binary
- 字符串最大长度：512 字节（可配置，但需重新编译）
- 单记录最大长度：1024 字节（可配置，但需重新编译）
- 最大记录条数：仅受存储空间限制
- 最大表的个数：仅受节点个数限制
- 最大数据备份数：5 份
- 单节点插入速度：10 万条/秒(单核，16 字节每记录，每次一条，无同步备份)
- 单节点查询速度：2000 万条/秒(单核，16 字节每记录，全内存)
- 更多指标将陆续提供

13 应用场景

TDengine 作为一个基础性的软件，应用范围及其广泛，原则上，所有使用机器、设备、传感器采集数据的地方都可以用上。一些典型场景罗列如下：

- 公共安全：上网记录、通话记录、个体追踪、区间筛选
- 电力行业：智能电表、电网、发电设备的集中监测
- 通讯行业：话费详单、用户行为、基站/通讯设备监测
- 金融行业：交易记录、存取记录、ATM、POS 机监测
- 出行工具：火车/汽车/出租/飞机/自行车的实时监测
- 交通行业：实时路况，路口流量监测，卡口数据；
- 石油石化：油井、运输管线、运输车队的实时监测
- 互联网： 服务器/应用监测、用户访问日志、广告点击日志
- 物流行业：车辆、集装箱的追踪监测
- 环境监测：天气、空气、水文、地质环境等监测；
- 物联网： 电梯、锅炉、机械、水表、气表等各种联网设备
- 军工行业：各种军事装备的数据采集、存储
- 制造业： 生产过程管控，流程数据、供应链数据采集与分析