Journal of Knowledge Information Technology and Systems(JKITS), Vol. 15, No. 5, pp. 671-681, October 2020
https://doi.org/10.34163/jkits.2020.15.5.010

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700 (Print), ISSN 2734-0570 (Online)

http://www.kkits.or.kr

Resource Sharing Scheme for Continuous Aggregation Query Processing in Spark Frameworks

Weonil Jeong*

Division of Computer and Information Engineering, Hoseo University

ABSTRACT

Recently, as the real-time data generated by sensors in connection with location information rapidly increases in the IoT environment, studies on various services using a big data platform have been conducted. In this big data platform, research on continuous query processing including data aggregation operation has been proposed to effectively analyze sensor big data constantly flowing for multi-user needs. However, the existing resource-sharing-based aggregation operation has a problem of concurrent execution of time-based and tuple-based queries and an increase in memory usage and deterioration in query processing performance due to duplicated maintenance of aggregate information. Therefore, in this paper, we propose a resource sharing method to effectively support continuous aggregation query processing based on Spark, an in-memory based distributed processing framework. The proposed method minimizes the increase in the cost of processing aggregate information through linear resource sharing based on summary information for the scope of query processing, and reduces the memory usage required for continuous aggregate query processing. Also, our method improves query processing performance by preventing duplicate generation of aggregate information. The proposed approach is implemented based on Spark framework to ensure real-time performance of continuous aggregate query processing for big data. Finally, through performance evaluation, it is shown that the proposed resource sharing technique can be effectively used for continuous aggregation query processing.

KEYWORDS: Spark framework, Continuous query, Aggregation, Continuous aggregation query, Resource sharing, Big data

ARTICLE INFO: Received 30 July 2020, Revised 25 August 2020, Accepted 13 October 2020.

*Corresponding author is with the Division of Computer and Information Engineering, Hoseo University, 79-20 Hoseo-ro Baebang-eup Asan, 31499, KOREA.

E-mail address: wnchung@hoseo.edu

1. 서 론

사물인터넷 환경에서 스마트 기기의 보급이 확산되고 소셜 미디어 등의 발달로 인해 다양한 정보 채널이 확대됨에 따라 이동기기의 위치 정보와 연계되어 유통되는 정보의 양이 기하급수적으로 증가하면서 빅데이터에 관심이 증가하고 있다[1-3]. 대규모 클러스터링 환경에서 작동하는 분산데이터 처리 프레임워크인 빅데이터 플랫폼에서는 사용자의 위치 정보나 시간 정보를 기반으로 대량의 센서 데이터 스트림에 대해 같은 질의 범위의 데이터 집계 연산을 수행하는 연속질의 처리기법들에 관한 연구가 제시되고 있다[4-7].

연속질의는 입력 순서에 따라 스트림 데이터를 처리하므로 갱신범위와 질의 범위로 구성된 슬라이딩 윈도우를 질의문에 포함하고 있으며[8]. 주요 데이터의 통계 산출을 위해 대부분 연속질의는 슬라이딩 윈도우 집계 연산을 포함한다. 이러한 연속 집계 질의는 슬라이딩 윈도우를 이동하면서 스트림 데이터를 연관 질의의 큐로 복사하여 처리하는데, 질의별 독립 큐를 유지하기 위한 메모리 비용이 증가한다. 이에 메모리 비용의 감소를 위해 B-Int 알고리즘, L-Int 알고리즘, 페인 등의 자원공유기법이 연구되었다[9-10]. B-Int 알고리즘은 질의 고유의 큐 메모리 공간을 할당할 필요는 없으나, 계층마다 집계정보의 유지 및 검색이 발생하므로 메모리 유지 비용의 증가 및 질의처리 성능 저하를 야기한다. L-Int 알고리즘은 대칭 집계 정보 관리를 통해 B-Int 알고리즘에 비해 집계정보 검색 성능은 우수하지만, 집계정보의 중복 유지로 인해 메모리 비용은 증가하게 된다. 페인 기반 방법은 질의 범위의 최대공약수 크기만큼의 집계정보를 저장하여 질의를 처리한다. 이는 동일 튜플이 질의에 의한 반복계산을 방지하여 집계정보 생성시간을 감소시키지만, 시간 또는 튜플 개수 단위로만 구조가 생성되므로 시간 또는 튜플 기반 질의가 동시에 하나의 버퍼에서 실행할 수 없다.

또한, 맵리듀스 기반의 기존 연속질의 처리기법에서는 입력 데이터가 지속해서 변화하므로 데이터 변경에 따른 그 결과의 갱신이 수행되어야 한다. 그러나 맵 단계와 리듀스 단계에서 데이터의 저장 및 검색을 위한 반복적인 디스크 접근은 프로세스 성능 및 시스템 이용률의 저하를 일으키므로 일괄처리 방식의 맵리듀스 프레임워크[2,11]에서는 연산 수행 결과의 실시간 응답성을 보장할 수 없다[3,12-15]. 맵리듀스 프레임워크의 실시간성을 보장하기 위한 연구로는 일괄처리 작업의 완료 시간을 감소시키고 시스템 이용률을 향상하기 위해 연산자들 사이에 데이터를 파이프라인으로 처리하는 맵리듀스 온라인[12], 맵리듀스 처리 과정에서 성능의 병목 현상을 일으키는 디스크 입출력 비용을 최소화하여 효과적인 데이터 분석을 지원하는 메인 메모리 기반의 범용적 분산 고성능 클러스터링 프레임워크로 스파크(Spark)[13-15]가 제시되었다. 스파크에서는 기본 데이터 단위로 RDD(Resilient Distributed Dataset)를 정의하고, 맵리듀스의 맵 단계와 리듀스 단계와 유사하게 데이터를 변형하는 Transform 단계와 집계 연산과 같은 연산을 수행하는 Action 단계로 나뉘어 데이터 정제과정을 거친다.

본 논문에서는 인-메모리 기반의 분산 처리 플랫폼인 스파크 프레임워크에서 실시간 빅데이터에 대해 효과적인 연속집계 질의처리를 위한 선형적 자원 공유기법을 제안한다. 제안 기법에서는 스파크의 Transform 단계와 Action 단계에서 각 튜플 및 시간 기반 질의 유형에 대해 질의 범위과 갱신 범위의 최대공약수로 페인(Pane)과 타임유닛(Time Unit) 단위의 크기를 결정하며, 집계정보는 페인 크기의 튜플들로 생성되고 타임유닛 단위로 이를 그룹핑하는 자원공유 기법을 활용한다[16]. 연속집계

질의처리 과정에서 질의 범위에 대해 튜플 기반 질의는 페인 정보로부터 집계정보를 산출하고, 시간 기반 질의는 타임유닛 단위로 관련 페인들에서 집계정보를 산출한다. 이러한 선형 구조를 이용하면 질의처리 시 입력 데이터에 대해 페인 크기의 집계정보를 한 번만 구성하게 하므로 집계정보 구축시간을 최소화할 수 있으며, 질의 범위 검색 시 집계정보를 한 번에 검색할 수 있어 질의처리 시간을 향상시킨다. 또한, 집계정보는 계층별 중복적 구축이 불필요하므로 메모리 소요비용이 감소하게 된다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구에 관해 기술하고, 3장에서는 집계정보에 대한 자원공유방법에 대해 논의하고, 4장에서는 자원공유기반 연속집계 질의처리 방안을 설명한다. 이어 5장에서 성능평가 결과를 제시한 후, 6장에서 결론을 맺고 향후 연구를 제시한다.

2. 관련 연구

2.1 자원공유기법

연속집계 질의처리를 위한 계층구조 형식의 자원공유 기법인 B-Int는 집계정보 생성 과정에서 튜플이 입력되면 최하위 계층에 저장하고, 입력된 두 튜플로부터 산출된 집계정보를 상위 계층에 유지한다. 이어 상위 계층의 두 집계정보로부터 하나의 집계정보를 계산하여 상위 계층에 저장한다. 이러한 과정으로 집계정보가 상위 계층에서 관리되는 계층구조를 이룬다. B-Int는 질의처리 과정에서 질의 범위를 만족하는 집계정보를 검색하고, 각 집계정보에 저장된 집계 값들을 계산하여 질의 결과를 반환한다. 이러한 B-Int 기법의 단점은 계층별 집계정보 유지를 위한 메모리 비용이 많고, 질의처리 시 질의 범위에 부합하는 집계정보를 탐색을 위해 계층별 검색이 발생하여 질의처리 성능이 저하된다는 문제가 있다.

계층구조 기반으로 B-Int의 단점을 개선하기 위해 제안된 L-Int은 계층구조를 좌우로 구축하여 두 범위만 검색하여 질의 결과를 생성한다. L-Int은 대칭 집계정보를 기반으로 질의 범위에 해당하는 계층의 집계정보를 이용하여 질의를 처리할 수 있다. 따라서 L-Int은 B-Int과 비교하면 질의 범위를 만족하는 대상을 검색하는 속도를 개선하였으나, 대칭형의 계층구조를 사용하기 때문에 계층구조 구축시간과 메모리 낭비는 B-Int보다 커지는 문제가 있다. 또한, 질의처리가 지연될 경우 메모리 비용이 증가하는 문제가 있다.

자원공유 기반의 연속집계 질의처리를 지원하는 페인 기법이 제시되었다. 연속집계 질의에서 질의 범위를 규정하는 슬라이딩 윈도우는 질의 범위가 갱신범위보다 크므로 질의처리를 위해 스트림 버퍼 탐색시 중복 영역을 포함하게 되는데, 페인 기법은 이러한 중복 영역의 반복계산을 방지한다. 페인 기법에서는 입력 데이터를 갱신범위와 질의 범위의 최대공약수 단위로 구분하여 튜플들의 집계값을 관리한다. 질의처리 과정에서는 분할 저장된 팬의 집계 값을 이용하여 질의 결과를 반환하므로 튜플 집계 연산은 페인 구조를 생성할 때 한 번만 수행된다. 이는 기존 자원공유 기법에서 동일 튜플에 대한 집계 연산이 중복으로 수행하는 것에 비해 검색 비용이 감소되어 질의처리 성능이 향상된다. 그러나 해당 페인 기법은 단일 질의에만 적용할 수 있으며, 튜플기반 질의와 시간기반 질의가 동시에 수행되는 환경에서는 페인의 크기를 결정할 수 없다. 또한, 갱신범위와 질의 범위가 서로 소이면 팬 크기를 결정할 수 없다. 본 논문에서는 이러한 페인 기법의 단점을 개선하여 연속집계 질의처리에서 유용한 구조로의 확장을 제시한다.

2.2 스파크 프레임워크

스파크는 인-메모리 기반의 오픈소스 클러스터 컴퓨팅 프레임워크로 빅데이터에 대한 고속의 분석 작업을 지원하며, 일반적인 구성은 아래 <그림 1>과 같다.

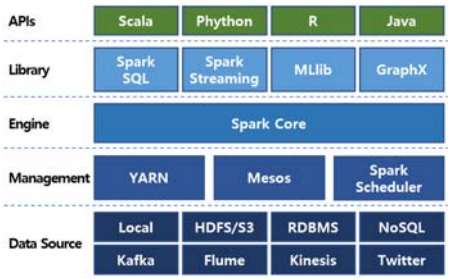

그림 1. 스파크 프레임워크
Figure 1. Spark framework

<그림 1>에서 스파크 프레임워크는 HDFS(Hadoop file system), RDBMS(Relational database management system), NoSQL 등 다양한 형태의 데이터 소스를 기반으로 동작한다. YARN, Mesos, Spark Scheduler 등의 관리 계층에서는 데이터 소스에 대한 리소스 할당, 작업 스케줄링 및 결함처리 기능을 제공한다. 스파크 프레임워크의 엔진(Engine) 계층의 스파크 코어(Spark Core)는 작업 스케줄 관리와 조정 및 RDD(Resilient Distributed Data)를 통해 빅데이터에 대한 다양한 연산 작업을 지원한다. Spark SQL, Spark Streaming, MLlib, GraphX 등 라이브러리(Library) 계층에서는 데이터 분석을 위한 용도별 라이브러리를 제공하며, 스파크 APIs 계층은 다양한 프로그래밍 개발을 위한 인터페이스를 제공한다.

스파크는 데이터 집합의 추상화 객체 개념으로 RDD를 제공하여, 다수의 노드에 분산하여 복수의 파티션으로 관리되어 장애가 발생해도 데이터 사용 이력 정보를 통해 복구할 수 있다. 또한, 스파크는 RDD를 분산 클러스터의 메모리에 저장하기 때문에 기계학습, 그래프 처리 등 반복계산이 많은 작업에 대해 하둡보다 실행 속도가 빠르다.

3. 집계정보 자원공유

제안 기법에서 집계정보의 구조는 튜플기반 질의를 위해 고정 튜플 개수 단위의 페인을 이용하고, 시간기반 질의를 위해 고정 시간단위의 타임유닛을 이용한다. 집계정보 목록은 타임유닛들이 리스트로 연결되며, 각 타임유닛은 다수의 페인을 포함한다. 타임유닛은 {시간값, 페인목록}으로 구성되며, 페인은 {집계값수, 페인유형, 집계정보목록} 구조로 집계정보를 저장한다.

3.1 집계정보 구조화

집계정보의 연속집계 질의처리 과정에서는 버퍼를 이동하면서 실행하는 범위에 포함되는 튜플들과 동일한 튜플로 구성된 페인과 타임유닛들을 검색한다. 이러한 페인과 타임유닛은 질의의 갱신범위와 질의범위를 고려한 크기로 설정되어야 하며, 등록된 연속질의들과 약수 관계여야 한다.

페인과 타임유닛이 질의 범위와 일치하면 약수 관계이며, <그림 2>에서 연속질의와 집계정보의 약수 관계에 대해 예시한다.

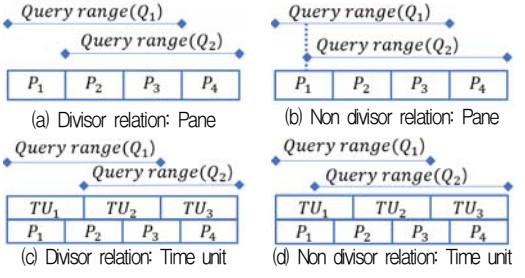

그림 2. 연속질의와 집계정보 약수 관계
Figure 2. Divisor relation between queries and aggregations

<그림 2>에서 연속질의와 페인 및 타임유닛과의 약수 관계를 4가지로 나타낸다. 연속질의 Q_1과 Q_2의 질의 범위에 대해 (a)는 Q_1과 Q_2가 약수 관계로 $\{P_2, P_3\}$을 공통범위를 가지며, (b)에서는 Q_2가 P_1의 중간 부분에 걸쳐있어 두 질의가 약수 관계에 있지 않다. 타임유닛은 다수의 페인을 포함할 수 있으며, (c)에서는 Q_1과 Q_2의 타임유닛이 약수 관계로 $\{TU_2\}$을 공통으로 포함하고 있으며, (d)는 Q_1과 Q_2에 대한 공통 타임유닛이 존재하지 않으므로 약수 관계에 있지 않음을 알 수 있다.

임의의 i번째 연속질의에 대해 갱신범위가 s_i, 질의 범위가 $r_i (r_i \geq s_i)$ 이고, m번째 실행되는 범위가 $I_{i,m} = (s_i m, s_i m + r_i)$일 때, 집계정보의 크기가 p이고 k부터 l까지 연속적으로 생성되는 집계정보의 범위의 합이 $I_{i,m}$과 같으면 페인과 연속질의는 약수 관계이다.

등록된 질의들의 질의 범위와 갱신범위의 최대공약수로 약수 관계를 산출하며, 그 관계가 서로소일 경우 집계정보의 크기는 1로 설정한다. 실험 환경에서 초당 입력률이 일 만개를 상회하므로 타임유닛 집계정보에서의 크기는 1초로 설정한다. 그러나 페인 크기는 튜플 개수가 단위이므로 그 크기가 1인 경우 튜플별 집계정보를 생성하므로 메모리 비용이 증가한다. 따라서 튜플 기반 질의들은 1보다 큰 약수 관계의 페인을 계산해야 한다.

튜플을 기반으로 실행되는 연속질의의 페인 크기를 산출하기 위해 연속질의의 갱신범위와 질의범위가 서로소가 아니면 NRP(Non relative prime) 질의, 서로소이면 RP(Relative prime) 질의로 구분한다. 이 두 유형의 질의 간의 관계를 $n (n \geq 1)$개 질의가 모두 NRP 유형이고, 갱신범위와 질의범위가 모두 서로소가 아닌 SP(Single Pane), n이 1인 단일 RP 유형인 DP(Double Pane), 2 이상의 n개 질의의 갱신범위와 질의범위가 하나라도 서로소의 경우인 GP(Group Pane)으로 정의한다.

SP는 서로소가 아닌 n개의 질의에 대해 갱신범위를 s_i, 질의범위를 r_i 라 할 때 약수 관계의 페인 크기는 $\gcd(s_1, s_2, ..., s_n, r_1, r_2, ..., r_n)$ 로부터 페인 크기를 결정한다. DP는 질의범위와 갱신범위의 최대공약수가 1이므로 이 값으로 페인을 구성하면 집계정보 유지 비용이 매우 크다. 이에 단일 서로소 관계의 질의에서 페인 크기는 갱신범위를 s, 1보다 큰 임의의 α에 대해 $(s\alpha + p)$인 r을 질의범위라고 할 때, 생성 페인의 크기는 질의와 약수관계인 2개의 p와 $(s - p)$를 활용한다. 이 두 개의 팬 크기를 사용하면 질의범위와 갱신범위가 서로소인 질의에서 갱신범위가 2보다 크면 1이 아닌 팬 크기를 사용하여 집계정보를 생성한다. GP는 하나의 페인 크기로 1보다 큰 약수 관계의 페인을 생성할 수 없으므로 질의들을 SP와 DP들의 그룹핑을 통해 1보다 큰 약수관계의 팬 크기를 결정한다.

3.2 집계정보 그룹화

그룹화는 질의를 특성별로 분류하여 다수의 그룹으로 분할하고, 각 그룹에 속한 질의들이 그룹별로 존재하는 집계정보 리스트를 공유하는 과정으로, 그룹화 기법에서의 그룹은 페인과 타임유닛을 동시에 생성하는 상위그룹(Upper group)과 페인만 생성하는 하위그룹(Lower group)으로 구분한다.

질의 관계에 따라 SP 유형의 상위그룹에 대한 그룹화 구조의 예시는 <그림 3>과 같다.

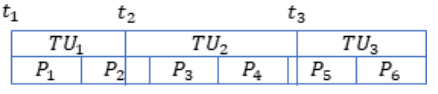

그림 3. SP 상위그룹 그룹핑

Figure 3. Grouping of Upper group for SP

<그림 3>은 상위그룹의 SP 그룹 구조의 예시로, 시간 흐름에 따라 타임유닛 및 페인의 관계를 나타낸다. 타임유닛은 한 개의 그룹으로 형성될 수 있으나, 페인은 여러 개의 그룹으로 분할 생성될 수 있으므로, 그룹은 타임유닛과 페인이 동시에 생성되는 상위그룹 한 개와 페인만 생성하는 다수의 하위그룹이 존재한다. 그리고 각 그룹은 모두 집계정보 리스트를 가지고 있고, 그룹별로 독립적인 페인 크기를 가진다.

4. 자원공유기반 연속집계 질의처리

본 논문에서 제안하는 연속집계 질의처리 기법은 <그림 4>와 같이 5계층으로 구성된 시스템을 기반으로 작동한다.

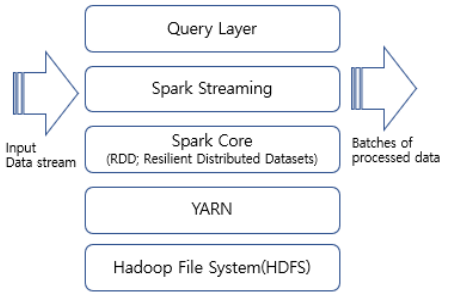

그림 4. 시스템 구조
Figure 4. System architecture

구조 데이터 저장을 위한 분산 파일 시스템인 HDSFS(Hadoop file system)과 분산 컴퓨팅 환경을 제공하는 YARN(Yet Another Resource Negotiator)로 구성된 하둡(Hadoop)을 기반으로 한다. Spark Core 계층에서는 RDD 데이터 구조를 이용하여 질의처리를 위한 데이터 변경 과정을 수행하고, Spark Streaming 계층에서는 실시간 대용량 데이터의 분산 처리를 지원하며, Query Layer에서는 연속집계 질의처리를 수행한다. 제안 기법에서 하둡 파일 시스템을 빅데이터 저장소로 활용하고, 스파크 스티리밍을 다양한 데이터 소스로부터 입력되는 실시간 스트림 데이터를 처리한다. 그리고, 연속집계 질의처리를 위한 RDD에 따른 데이터 변형 작업은 트랜스폼 단계에서 수행하고, 질의처리는 액션 단계에서 수행하여 결과를 반환한다.

4.1 집계정보 구성

페인 집계정보는 약수 관계의 페인 크기 및 타임유닛의 크기를 기준으로 생성한다. 페인 생성은 입력 스트림 데이터의 튜플들이 페인 크기에 해당하는 완전 상태와 페인 크기와 일치하지는 않으나 타임유닛의 시간 제약을 초과하는 불완전 상태로 구분되고, 집계정보 생성 과정에서의 페인은 임시 메모리에 유지되며 모든 페인들의 집합은 하나의 타임유닛으로 관리된다.

<그림 5>는 t_2부터 t_3까지 타임유닛과 페인의 생성 과정을 나타낸다.

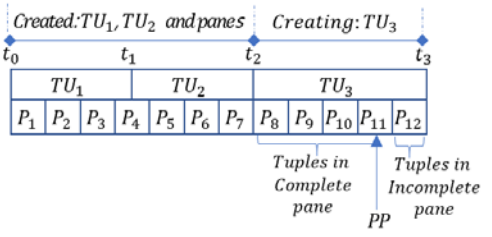

그림 5. 타임유닛과 페인 생성 과정
Figure 5. Creation steps of time unit and pane

버퍼에서 PP(Pointer position)가 위치까지 집계정보를 임시메모리에 저장한다. $\{P_8, P_9, P_{10}, P_{11}\}$는 완전한 상태의 페인으로 생성되었고 P_{12}는 생성되고 있다. 이후 P_{12} 생성과 함께 타임유닛의 종료시간인 t_3 튜플이 입력되면 불완전 상태인 P_{12}이 생성되고, $\{P_8, P_9, P_{10}, P_{11}, P_{12}\}$로 구성된

타임유닛 TU_3가 집계정보 리스트에 저장된다. 이후 페인은 계속 버퍼에서 튜플을 읽어서 집계 값을 저장하고, 최종 페인 생성이 완료될 때까지 생성된 집계 값을 페인에 저장한다. 연속집계 질의에서 집계 연산은 최소(MIN), 최대(MAX), 개수(COUNT), 합계(SUM), 평균(AVG)을 지원한다. 다른 집계 연산과 달리 평균(AVG)은 완전한 상태와 불완전한 상태의 페인은 구성 튜플 수가 상이하므로 개수(COUNT)와 합계(SUM)으로 나누어 저장함으로써 집계정보의 정확도를 유지한다.

4.2 집계정보 탐색 처리

질의처리 과정에서 집계정보의 탐색은 튜플기반 질의는 질의범위가 튜플 단위이므로 저장된 페인을 기준으로 결과를 반환하고, 시간기반 질의는 질의범위가 시간 단위이므로 저장된 타임유닛을 기준으로 결과를 반환한다.

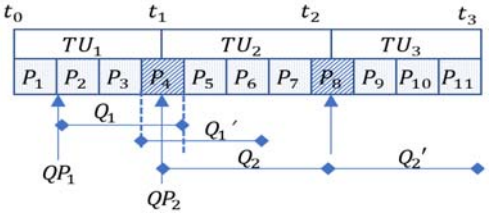

그림 6. 타임유닛과 페인의 탐색 과정

Figure 6. Searching steps of time unit and pane

<그림 6>은 튜플기반 Q_1 질의와 시간기반 Q_2 질의가 동시에 실행되는 상황에서 페인과 타임유닛을 탐색하는 과정을 나타낸다. Q_1 질의범위는 페인 크기가 3으로, 여기에는 완전 상태인 P_2와 P_3, 불완전 상태로 t_1 시점에서 P_4'와 P_4''로 나뉘는 P_4를 포함하며, $\{P_2, P_3, P_4', P_4''\}$에 저장된 집계정보로부터 결과를 처리한다. Q_2 질의는 타임유닛 t 단위로 QP를 이동하며 질의처리를 수행하며, 질의범위에는 $\{P_4'', P_5, P_6, P_7, P_8'\}$에 해당한다.

4.3 집계정보 후처리

집계정보 목록에서 사용되지 않는 페인들은 삭제하며, 이를 위해 질의처리 후 가장 후행하는 QP를 LP로 설정하고 LP가 이전 타임유닛의 위치에서 다음 타임유닛으로 이동할 경우 이전 위치의 타임유닛을 삭제하여 페인 버퍼에서 삭제한다.

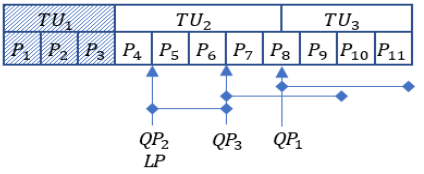

그림 7. 집계정보 목록에서 페인 삭제

Figure 7. Deleting panes from aggregation list

<그림 7>에서 페인 버퍼에 존재하는 질의 포인터는 $\{QP_1, QP_2, QP_3\}$이며, 가장 후행하는 QP_2가 LP로 설정된다. 이에 TU_1에 포함되는 $\{P_1, P_2, P_3\}$가 삭제 대상이 된다.

5. 성능 평가

5.1 실험 환경

실험 환경은 우분투 18.04 64비트 운영체제, Intel Xeon 3.5Ghz CPU, 128GB 메모리로 구성된 시스템에 아파치 스파크 3.0.0을 구성하였다. 실험에 사용된 입력 스트림 데이터는 초당 최대 10만 개의 튜플을 임의로 생성하였다. 튜플의 구조는 $\langle RegionID, ObjectID, LocX, LocY, TStamp\rangle$로 20바이트 크기로 설정하였으며, 집계 처리를 위해 $RegionID$는 16개 영역으로 분할 처리한다.

5.2 성능 분석

실험에서는 제안 자원공유 기법(RS)과 기존의 B-Int, L-Int를 비교 평가하였다.

첫 번째 실험은 단일 튜플 기반 질의에 대해 입력 튜플 수의 증가에 따른 집계정보 생성시간 변화를 분석하였다.

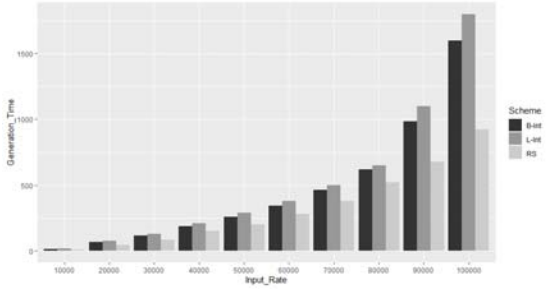

그림 8. 입력 튜플 수 증가에 따른 집계정보 생성시간 분석
Figure 8. Analysis of the generation time of aggregation information to input rate

<그림 8>은 입력 튜플 증가에 따른 집계정보 생성시간을 비교하였다. 제안 기법(RS)에서 단일 튜플은 여러 계층이 아닌 하나의 페인에만 속하므로, 튜플의 중복 연산이 불필요하다. 결과에서 B-Int 및 L-Int가 제안 기법보다 전체 평균 36%, 50% 수준의 집계정보 생성시간이 더 소요되고 있다.

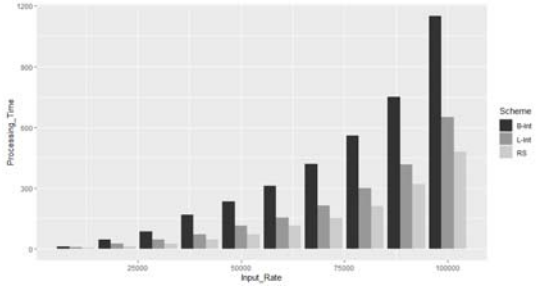

그림 9. 입력 튜플 수 증가에 따른 질의 처리시간 분석
Figure 9. Analysis of the query processing time to input rate

두 번째 실험은 단일 질의를 대상으로 입력 튜플 수의 증가가 질의처리 소요시간을 분석하였다.

<그림 9>에서 제안 기법(RS)은 질의 범위에 해당하는 집계정보만을 검색하여 질의처리를 수행하므로 B-Int와 L-Int에 비해 월등하게 우수한 성능을 보여주고 있다.

세 번째 실험은 시간과 튜플 기반 질의가 동시에 수행되는 경우 시간 및 튜플 기반 질의의 비율을 변화시키면서 질의처리 속도를 분석하였다. 실험에는 시간 및 튜플 기반 연속질의 20개의 비율을 조정하여 사용하였고, 입력 튜플은 초당 일 만개로 설정하였다. B-Int와 L-Int는 시간 및 튜플 기반 질의를 동시 수행할 수 없어, 각 기법에서 튜플 기반 질의는 고유의 방법을 적용하고 시간 기반 질의에는 자원을 공유하지 않는 방법을 이용하였다.

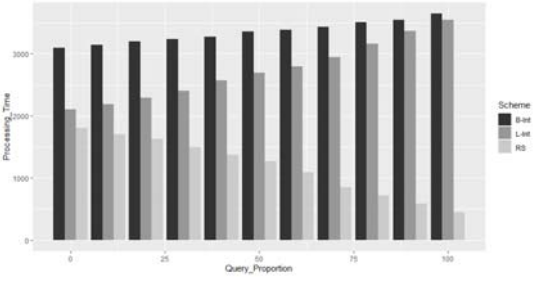

그림 10. 질의 비율 변화에 따른 질의처리시간 분석
Figure 10. Analysis of the query processing time to query proportion

<그림 10>에서 질의 비율은 튜플 기반 질의 100%로부터 시간 기반 질의를 10%씩 증가하면서 질의처리 속도의 결과를 나타낸다. 스트림 데이터의 입력이 증가하면 튜플 기반 질의가 시간 기반 질의보다 질의처리 빈도가 높아지므로 실험에서 B-Int와 L-Int의 질의처리 속도가 전반적으로 크다는 것을 알 수 있다. 시간 기반 질의 비율을 증가할수록 튜플의 중복 계산을 방지하는 제안 기법

(RS)의 질의처리 속도가 더욱 효과적임을 알 수 있다.

네 번째 실험에서는 연속질의의 수가 증가할 때 자원 사용량의 효과성을 검증한다. NRP 유형의 연속질의를 80%, RP 유형의 연속질의를 20%로 구성하여 10개씩 증가시켰으며, 입력 데이터는 초당 일만개를 사용하여 튜플 입력이 완료된 시점에서 할당된 메모리 사용량을 분석하였다.

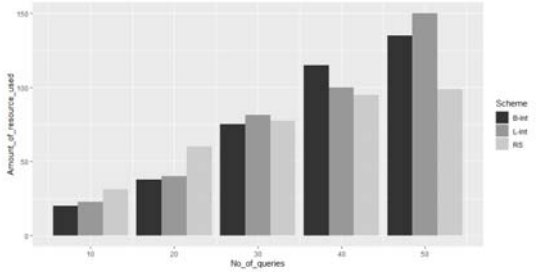

그림 11. 질의 수 변화에 따른 메모리 사용량 분석
Figure 11. Analysis of the amount of resources used to no. of queries

<그림 11>에서 수행 질의 수의 변화에 비례하여 B-Int와 L-Int의 메모리 사용량은 증가하고 있다. 제안 기법(RS)은 초기 질의 수 증가에 따라 그룹 수 증대로 다른 기법에 비해 메모리 사용량이 높은 수준을 보이나, 질의 수가 증가할수록 그룹과 약수 관계일 확률이 높아져 질의 수가 20개에서 30개 사이 구간을 기점으로 메모리 사용량의 증가가 둔화되고 있다. 또한, 제안 기법은 앞선 실험에서 검증한 결과와 같이 다른 기법에 비해 상대적으로 질의처리 속도가 우수하므로 질의처리에 사용된 메모리 회수가 빠르게 진행되므로 질의 수 증가에 따른 메모리 사용량이 완화된 것을 알 수 있다.

6. 결 론

본 논문은 인-메모리 기반 분산 처리 플랫폼인 스파크 프레임워크에서 질의처리 속도와 메모리 사용률을 증대시키기 위해 자원공유를 이용한 연속집계 질의처리 기법을 제안하였다. 제안 기법은 페인과 타임유닛 구조를 기반으로 질의집계정보 그룹핑 과정을 통해 튜플 및 시간기반 질의의 동시 수행을 지원할 뿐 아니라 질의범위와 갱신범위 사이의 페인 크기 결정 문제를 해소하였다.

실험에서 질의처리 시간에 영향을 미치는 집계정보 생성시간, 입력 튜플 수와 질의 비율의 변화에 따른 질의처리시간 결과에서 제안 기법은 기존 계층형 자원공유 기법과 비교하여 우수한 성능을 보였다. 또한, 질의 수 변화에 따른 메모리 사용량 분석에서는 기존 기법들이 선형적인 메모리 사용량 증가 추이를 보였으나, 제안 기법은 질의 수 증가에 그룹 수 증가가 정비례하지 않고 빠른 질의처리 속도 및 메모리 회수로 인해 메모리 사용량의 증가 속도가 급격히 떨어지는 결과를 보였다.

향후 연구로는 하둡에 저장된 디스크 기반 데이터와 연계한 효과적인 연속집계 질의처리 방법과 질의 유형에 따라 질의처리의 우선순위 부여에 관한 연구가 필요하다.

References

[1] M. Beyer, *Gartner says solving 'Big Data' challenge involves more than just managing volumes of data*, Gartner, Jun. 27, 2011.

[2] J. Dean, and S. Ghemawat, *MapReduce: Simplified data processing on large clusters*, Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pp. 137-150, 2004.

[3] W. Jeong, *Dynamic load management method for spatial data stream processing on MapReduce online frameworks*, Journal of the

Korea Academia-Industrial cooperation Society, Vol. 19, No. 8, pp.535-544, 2018.

[4] A. M. Aly, A. Sallam, B. M. Gnanasekaran, L. Nguyen-Dinh, W. G. Aref, M. Ouzzani, and A. Ghafoor, *M3: stream processing on main-memory MapReduce*, Data Engineering (ICDE), IEEE 28th International Conference, pp. 1253-1256, 2012.

[5] D. Jeong, S. Jeon, and B. Hong, *A study on MapReduce processing for multi-dimensional continuous query*, Lecture notes in computer science, Vol. 7827, pp. 74-78, 2013.

[6] D. Yang, J. Cao, S. Wu, and J. Wang, *Progressive online aggregation in a distributed stream system*, Journal of systems and software, Vol. 102, pp. 146-157, 2015.

[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, *Online aggregation and continuous query support in MapReduce*, Special Interest Group on Management of Data 2010, pp. 6-11, Jun. 2010.

[8] J. Li, and D. Maier, *Semantics and evaluation techniques for window aggregates in data streams*, Proceedings of ACM SIGMOD, pp. 311-322, 2005.

[9] A. Arasu, and J. Widom, *Resource sharing in continuous sliding-window aggregates*, Proceedings of the Very Large DataBase, pp. 336-347, 2004.

[10] J. Li, and D. Maier, *No Pane, No Gain : Efficient evaluation of sliding-window aggregates over data streams*, SIGMOD Record, pp. 39-44, 2005.

[11] Apache Hadoop, https://hadoop.apache.org/, Feb. 2020.

[12] T. Condie, N. Conway, P. ALvaro, and J. M. Hellerstein, *MapReduce online*, Networked Systems Design and Implementation, 2010.

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*, Proceedings of the 9th USENIX Symposium on networked systems design and implementation, USENIX Association, pp. 15-28, 2012.

[14] A. Gounaris, G. Kougka, R. Tous, C. T. Montes, and J. Torres, *Dynamic configuration of partitioning in spark applications*, IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No. 7, Jul. 2017.

[15] Apache Spark, https://spark.apache.org/, Feb. 2020.

[16] S-H. Back, B-S. You, S-K. Cho, and H-Y. Bae, *Linear resource sharing method for query optimization of sliding window aggregates in multiple continuous queries*, Journal of KIISE, Database Vol. 33, No. 6, pp. 563-577, 2006.

스파크 프레임워크에서 연속집계 질의처리를 위한 자원공유기법

정원일

호서대학교 컴퓨터정보공학부 교수

요 약

최근 사물인터넷 환경에서 센서가 위치 정보와 연계하여 생성하는 실시간 데이터가 급속하게 증가함에 따라 빅데이터 플랫폼을 활용한 다양한 서비스에 관한 연구들이 수행되고 있다. 이러한 빅데이터 플랫폼에서는 다중 사용자 요구에 대해 끊임없이 유입되는 센서 빅데이터를 효과적으로 분석하기 위해 데이터 집계 연산을 포함한 연속질의 처리에 관한 연구가 제시되어왔다. 그러나 기존의 자원공유 기반의 집계연산

은 시간기반 질의와 튜플기반 질의의 동시 수행의 문제와 집계 정보의 중복 유지로 인한 메모리 사용량 증가 및 질의처리 성능 저하가 발생하고 있다. 이에 본 논문에서는 인-메모리 기반 분산 처리 프레임워크인 스파크를 기반으로 연속집계 질의처리를 효과적으로 지원하기 위한 자원 공유기법을 제안한다. 제안 기법은 질의처리 범위에 대해 집계정보 기반의 선형 자원공유를 통해 집계 정보 처리 비용의 증가를 최소화하고, 집계정보의 중복 생성을 방지함으로써 연속집계 질의처리에 요구되는 메모리 자원 사용량을 최소화하고 질의처리 성능도 향상시킨다. 또한, 제안 기법은 빅데이터에 대한 연속집계 질의처리의 실시간성을 보장하기 위해 스파크 프레임워크를 기반으로 구현한다. 마지막으로 성능 평가를 통하여 제안 자원 공유기법이 연속집계 질의처리에 효과적으로 활용될 수 있음을 보인다.

감사의 글

"이 논문은 2018년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임"(20180353)

Weonil Jeong received the Ph.D. degree in the Department of Computer Science & Engineering from Inha University in 2004. He has been a professor in the Division of Computer & Information Engineering at Hoseo University since 2007. His current research interests include big data, system security, cloud computing.

E-mail address: wnchung@hoseo.edu