KIPS Tr. Software and Data Eng.
Vol.7, No.5 pp.177~188 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.5.177

Dynamic Block Reassignment for Load Balancing of Block Centric Graph Processing Systems

Yewon Kim† · Minho Bae†† · Sangyoon Oh†††

ABSTRACT

The scale of graph data has been increased rapidly because of the growth of mobile Internet applications and the proliferation of social network services. This brings upon the imminent necessity of efficient distributed and parallel graph processing approach since the size of these large-scale graphs are easily over a capacity of a single machine. Currently, there are two popular parallel graph processing approaches, vertex-centric graph processing and block centric processing. While a vertex-centric graph processing approach can easily be applied to the parallel processing system, a block-centric graph processing approach is proposed to compensate the drawbacks of the vertex-centric approach. In these systems, the initial quality of graph partition affects to the overall performance significantly. However, it is a very difficult problem to divide the graph into optimal states at the initial phase. Thus, several dynamic load balancing techniques have been studied that suggest the progressive partitioning during the graph processing time. In this paper, we present a load balancing algorithms for the block-centric graph processing approach where most of dynamic load balancing techniques are focused on vertex-centric systems. Our proposed algorithm focus on an improvement of the graph partition quality by dynamically reassigning blocks in runtime, and suggests block split strategy for escaping local optimum solution.

Keywords : Block-Centric Processing, Large-Scale Graphs, Load Balancing, Block Reassignment

블록 중심 그래프 처리 시스템의 부하 분산을 위한 동적 블록 재배치 기법

김 예 원† · 배 민 호†† · 오 상 윤†††

요 약

최근 웹, 소셜 네트워크 서비스, 모바일, 사물인터넷 등의 ICT 기술의 발전으로 인해 처리 및 분석이 필요한 그래프 데이터의 규모가 급속하게 증가하였다. 이러한 대규모 그래프 데이터는 단일 기기에서의 처리가 어렵기 때문에 여러 기기에 나누어 분산/병렬 처리하는 것이 필요하다. 기존 그래프 처리 알고리즘들은 단일 메모리 환경을 기반으로 연구되어 분산/병렬 처리환경에 적용되기 힘들다. 이에 대규모 그래프의 보다 효과적인 분산/병렬 처리를 위해 정점 중심 방식의 그래프 처리 시스템들과, 정점 중심 방식의 단점을 보완한 블록 중심 방식의 그래프 처리 시스템들이 등장하였다. 이러한 시스템들은 초기 그래프 분할 상태가 전체 처리 성능에 상당한 영향을 미친다. 한 번에 최적의 상태로 그래프를 분할하는 것은 매우 어려운 문제이므로, 그래프 처리 시간에 점진적으로 그래프 분할 상태를 개선하는 여러 로드 밸런싱 기법들이 연구되었다. 그러나 기존 기법들은 대부분 정점 중심 그래프 처리 시스템을 대상으로 하여 블록 중심 그래프 처리 시스템에 적용이 어렵다. 본 논문에서는 블록 중심 그래프 처리 시스템을 대상으로 적용 가능한 로드 밸런싱 기법을 제안한다. 제안 기법은 동적으로 블록을 재배치하여 점진적으로 그래프 분할 상태를 개선시키며, 해를 찾아나가는 과정에서 지역 최적해를 벗어나기 위한 블록 분할 전략을 함께 제시한다.

키워드 : 블록 중심 처리, 대규모 그래프 데이터, 로드 밸런싱, 블록 재배치

1. 서 론

최근 모바일 기기, 소셜 네트워크 서비스, 웹, 사물인터넷, 위치기반 기술 등의 급속한 발전으로 인해 정점과 간선의 수가 매우 큰 규모를 가지는 그래프 데이터가 증가하였다. 일반적으로 이러한 대규모 그래프 데이터는 단일 노드의 수용량을 쉽게 초과하여 여러 기기에 나누어 분산/병렬 처리하는 것이 필요하다. 그러나 그래프 데이터의 분산/병렬 처리는 그래프가 가지는 연결특성(interdependency)으로 인해 단일 노드의 공유메모리 환경을 기반으로 연구된 기존 그래프 처리 방법을 적용하기 어렵다. 기존 그래프 처리 방법들은 단일 노드 환경에서 전체 그래프 데이터가 공유메모리를 통해 서로 임의접근(random access)이 가능하다고 전제한다. 따라서 이

※ 이 논문은 2017년도 정부(교육부)재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015R1D1A1A01059557).
† 비 회 원 : 리디북스 사원
†† 비 회 원 : 아주대학교 컴퓨터공학과 석 · 박통합과정
††† 종신회원 : 아주대학교 소프트웨어학과 교수
Manuscript Received : December 15, 2017
First Revision : February 20, 2018
Accepted : March 13, 2018
* Corresponding Author : Sangyoon Oh(syoh@ajou.ac.kr)

를 단순히 분산/병렬 처리하는 경우 분산 노드에 나누어진 데이터에 대규모의 임의접근이 발생, 처리 성능이 매우 낮아지는 문제가 발생한다[1].

빅데이터의 중요성이 인식되기 시작한 초기에는 보편적으로 널리 사용되는 하둡[2] 프레임워크를 기반으로 그래프를 처리하는 연구들이 제안 되었다[3, 4]. 그러나 하둡은 맵리듀스[5] 방식으로 데이터를 처리하는 과정에서 디스크 접근을 필요로 한다. 이러한 특성은 그래프 알고리즘의 분산/병렬 처리 시 발생하는 임의 접근에 대해 대규모의 통신과 디스크 접근이 발생하는 요인이 되며, 결과적으로 그래프 처리 성능에 한계점으로 작용한다.

이와 같이 그래프의 분산 처리 시 임의접근으로 인한 대규모 통신 문제 해결을 위해 Pregel[1], GraphLab[6], PowerGraph[7], GraphX[8], GPS[9], Mizan[10]과 같은 정점 중심(vertex-centric) 그래프 처리 시스템들이 등장하였다. 정점 중심 그래프 처리 시스템들은 그래프 데이터를 분할하여 분산 메모리에 올려 그래프 처리 작업을 수행하는 방식으로 기존 하둡 맵리듀스의 디스크 접근에 의한 성능 문제를 해결하였다. 또한, 정점 중심 그래프 계산 모델을 적용하여 그래프 내 정점에 대한 임의접근으로 인한 대규모 통신 발생 문제를 해결하였다. 정점 중심 그래프 계산 모델에서는 하나의 정점의 대한 지역적(local) 정보에만 접근이 가능한 사용자 정의 함수(user-defined function)가 각 정점별로 여러 반복단계를 거쳐 분산/병렬 처리된다. 이러한 정점 중심 그래프 처리 시스템들은 대부분 BSP(Bulk Synchronous Parallel)[11] 방식으로 구현되어 전체 작업이 여러 슈퍼스텝(superstep)들의 연속으로 구성되고, 정점과 정점간의 통신이 메시지 전달(message passing)을 통해 수행된다.

그러나 이러한 메시지 전달은 같은 분할 그래프에 속한 정점 간에도 반드시 요구되어 불필요한 통신이 발생하고, 정점 사이의 거리가 길수록 메시지 전달을 위해 더 많은 슈퍼스텝이 소요된다. 따라서 그래프 데이터의 평균적인 지름(diameter)이 클수록 데이터 전파 속도가 느려지며, 그래프 알고리즘이 수렴 상태에 도달하기까지의 시간이 증가한다.

최근에는 이러한 문제를 해결하기 위해 Giraph++[12], GoFFish[13], Blogel[14], BLADYG[15] 등의 블록 중심(block-centric) 그래프 처리 시스템들이 제안되었다. 블록 중심 그래프 처리 시스템들은 블록을 연결 부분 그래프(connected subgraph)로 정의하여 동일 블록 내 정점들이 메시지 전달 과정 없이 메모리를 통해 접근할 수 있도록 하였다. 따라서 동일 블록 내 데이터 전파가 하나의 슈퍼스텝에 이루어지며, 결과적으로 전체 슈퍼스텝 수와 그래프 처리 시간이 감소한다. 블록 중심 방식은 정점 중심 방식의 단점을 개선하는 방법으로서의 큰 가능성을 가지고 있으나, 블록을 생성하는 단계와 블록의 크기에 따라 전체 처리 성능이 차이가 발생하는 문제가 있다.

블록 중심 그래프 처리 시스템뿐만 아니라, 대부분의 그래프 분산/병렬 처리 시스템에서는 초기 그래프 분할 상태에 따라 처리 성능이 크게 좌우된다. 일반적으로 분할되는 크기가 균일(balanced)하지 않으면 노드 간 부하 불균형이 발생하며, 분할된 그래프 간에 걸쳐진 절단 간선(edge-cut) 수가 많을수록 전반적인 통신비용이 증가한다. 따라서 절단 간선 수를 최소화하고 분할 그래프의 크기가 균일하도록 잘 분할하는 것이 중요하다. 이는 잘 알려진 NP-Complete 문제[16]로 이와 관련하여 많은 연구들[17-19]이 진행되었으나, 기존 방법들은 단시간 내 정확한 해를 구하는 것이 매우 어렵다.

최근에는 한 번에 정확한 해를 구하는 대신 그래프 처리 중 동적으로 그래프 분할 상태를 조금씩 개선하는 로드 밸런싱(load balancing) 기법들이 연구되었다[9, 10, 20, 21]. 그러나 대부분 정점 중심 그래프 처리 시스템을 대상으로 연구되어 제안 기법들이 정점 중심 계산 모델을 기반으로 동작한다. 기존 동작 방식을 블록 중심 계산 모델 기반으로 수정한다 하더라도, 기존 기법들이 블록의 연결성(inner connectivity)을 고려하지 않아 블록 중심 그래프 처리 시스템에 바로 적용되기 어렵다. 따라서 블록의 연결성을 고려하여 블록 중심 그래프 처리 시스템의 그래프 분할 상태를 동적으로 개선하는 로드 밸런싱 기법의 연구가 필요하다.

본 논문에서는 블록 중심 그래프 처리 시스템의 로드 밸런싱을 위한 동적 블록 재배치 기법을 제안한다. 본 논문의 제안 기법은 동적 블록 재배치를 통해 분할 그래프 간의 절단 간선 수를 감소시켜 한 번에 최적해를 구하는 대신 블록 재배치를 통해 점진적으로 해를 개선해 나가는 방식을 취한다. 블록이 재배치의 단위가 되기 때문에 블록의 연결성을 유지하여 블록 중심 그래프 처리 시스템이 가지는 이점을 극대화하며, 필요한 경우 블록 분할을 수행하여 지역 최적해(local optimum solution)에 빠지는 것을 방지한다.

본 논문의 공헌은 다음과 같다. 1) 블록 중심 그래프 처리 시스템에서 동적으로 블록을 재배치하여 그래프 분할 상태를 개선하는 방법을 제안하였다. 2) 블록 재배치 수행을 통해 해를 찾아나가는 과정에서 지역 최적해를 벗어나기 위한 블록 분할 전략을 제시하였다. 3) 본 제안 기법의 성능 평가를 위해 기존 블록 기반 시스템의 핵심적인 기능을 가지는 시스템을 구현하고 성능 평가를 진행하였다.

본 논문의 구성은 다음과 같다. 2장에서는 정점 중심 그래프 처리 시스템과 블록 중심 그래프 처리 시스템의 전반적인 구조와 동작 방식을 살펴보고, 3장에서는 정점 중심 그래프 처리 시스템의 동적 로드 밸런싱을 위한 기존 연구 기법들을 소개한다. 4장에서는 본 논문에서 제안하는 동적 블록 재배치 기법의 전반적인 처리 구조와 구체적인 블록 재배치 방법 및 블록 분할 방법에 대해 자세히 다룬다. 5장에서는 제안한 방법의 성능 및 측정 방법과 결과에 대한 분석을 설명한다. 마지막 6장에서는 본 논문의 결론을 내리고 향후 연구방향을 제시한다.

2. 배 경

본 장에서는 정점 중심 그래프 처리 시스템과 블록 중심 그래프 처리 시스템의 전반적인 구조와 동작 방식에 대해 설명하고, 블록 중심 그래프 처리 시스템이 정점 중심 그래프 처리 시스템에 비해 가질 수 있는 장점을 살펴본다.

대부분의 정점 중심 그래프 처리 시스템과 블록 중심 그래프 처리 시스템은 BSP를 기반으로 한 Master-Worker 구조를 가지며, 계산 작업이 여러 슈퍼스텝의 연속으로 구성된다. 하나의 슈퍼스텝 동안에는 각 컴포넌트가 내부 계산 작업을 수행하며, 계산 수행을 위해 다른 컴포넌트와 네트워크를 통해 메시지를 교환할 수 있다. 컴포넌트들은 작업을 마치면 다음 슈퍼스텝이 시작되기까지 대기하며, 모든 컴포넌트들의 작업이 끝나면 동기화가 이루어지고 다음 슈퍼스텝이 시작된다. Fig. 1은 이러한 BSP 방식의 처리 구조를 나타낸다.

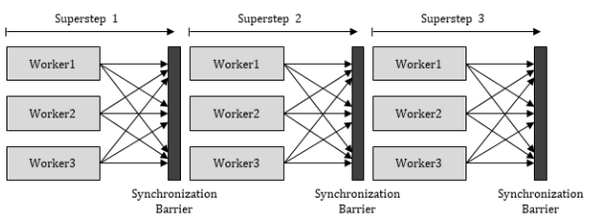

Fig. 1. BSP Architecture

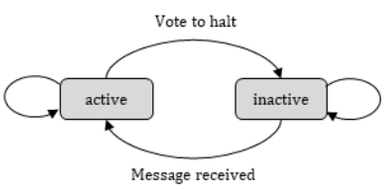

Fig. 2. Vertex State Machine

2.1 정점 중심 그래프 처리 시스템 동작 구조

Pregel, Giraph, GPS, Mizan과 같은 정점 중심 그래프 처리 시스템에서는 앞서 설명한 BSP 방식의 Master-Worker 구조를 기반으로 각 정점에 대해 사용자 정의 함수를 수행한다. 사용자 정의 함수는 하나의 정점에 대해 정점의 식별자, 정점의 이웃 정점들에 대한 정보, 정점이 받은 메시지 등에 접근이 가능하며, 진행되고 있는 슈퍼스텝의 단계 값 및 전역으로 관리되는 변수에 접근이 가능하다. 각 정점은 활성/비활성 상태를 가지며 활성 상태인 정점들에 대해서만 사용자 정의 함수가 수행된다. 사용자 정의 함수에서 특정 경우에 정점의 상태를 다음 슈퍼스텝에 비활성 상태로 변경하는 것이 가능하며, 비활성 상태의 정점은 다음 슈퍼스텝에 메시지를 받게 되면 다시 활성 상태로 변경된다. 전체 계산 작업은 모든 정점들의 상태가 비활성화 상태이고 다음 슈퍼스텝에 전달되는 메시지가 없는 경우 종료된다. Fig. 2는 이러한 정점의 상태 기계를 나타낸다.

2.2 블록 중심 그래프 처리 시스템 동작 구조

웹 그래프, 소셜 그래프 등 실세계 그래프(real-world graph)들은 정점의 평균 차수(degree)가 전반적으로 높고 정점의 차수 분포가 고르지 않고 치우치는 경향을 보이며, 굉장히 큰 길이의 지름을 가지는 것이 일반적이다[14]. 그러나 정점 중심 그래프 처리 시스템의 경우 이러한 실세계 그래프들을 처리 시 네트워크 오버헤드와 데이터 전파 속도 저하 문제가 발생한다. 정점 중심 그래프 처리 시스템의 정점 간 메시지 전달은 정점이 속한 분할 그래프를 고려하지 않고 수행되어 동일한 분할 그래프 내의 정점 간에 메시지를 보낼 때에도 네트워크 통신이 발생하며, 전달되는 메시지는 반드시 다음 슈퍼스텝에서 처리되는 구조를 가진다. 이는 평균적인 지름이 큰 그래프를 처리하는 경우 데이터가 전파되는 속도가 느려지고 많은 수의 슈퍼스텝이 소요되어 전반적인 그래프 계산 작업이 느려지는 요인이 된다[14].

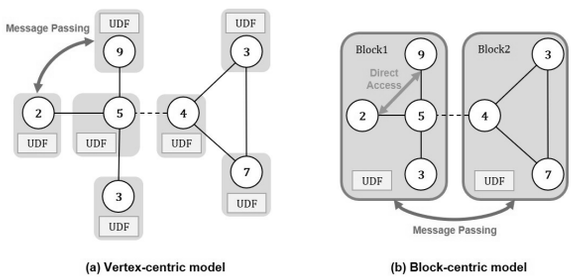

Fig. 3. Execution Range of UDF in Vertex-Centric and Block-Centric Model

Giraph++[12], GoFFish[13], Blogel[14], BLADYG[15] 등의 연구에서는 이러한 정점 중심 그래프 처리 시스템의 문제점을 해결하기 위해 블록(block) 혹은 부분 그래프(subgraph)를 하나의 처리 단위로 하는 계산 모델을 제안하였다. 본 논문에서는 이러한 계산 모델을 블록 중심 계산 모델로 지칭하고, 이러한 계산 모델을 사용하는 시스템들을 블록 중심 그래프 처리 시스템으로 통일하여 설명한다.

블록 중심 그래프 처리 시스템들은 그래프 처리의 단위가 하나의 정점이 아닌 하나의 블록이 된다. 블록은 여러 정점들의 집합으로 구성되며, 블록 내 정점들은 모두 서로 연결된 부분 그래프로 정의된다. 블록은 자신이 가지는 정점들에 대한 데이터와 블록 데이터 값, 블록 활성 혹은 비활성 상태를 가진다. 일반적으로 이러한 블록은 여러 기기로 나뉜 분할 그래프 내의 연결된 부분 그래프로부터 생성된다. 블록 중심 그래프 계산 모델은 기존의 정점 중심 그래프 계산 모델과 유사하게 사용자 정의 함수가 각 블록별로 여러 반복단계를 거쳐 분산/병렬 처리된다. 사용자 정의 함수는 하나의 블록을 접근 범위로 하여 동일 블록 내의 모든 정점에 접근이 가능하며, 블록 간의 통신이 메시지 전달 방식으로 이루어진다.

Fig. 3은 이러한 정점 중심 계산 모델과 블록 중심 계산 모델의 사용자 정의 함수(UDF) 실행 범위를 보여 준다 정점 중심 계산 모델에서 사용자 정의 함수는 각 정점에 대해 개별적으로 병렬 수행되며, 각 사용자 정의 함수는 자신이 담당하는 하나의 정점에만 접근이 가능하다. 정점과 정점간의 통신은 메시지 전달 방식으로 수행된다. 블록 중심 계산 모델에서 사용자 정의 함수는 각 블록에 대해 개별적으로 병렬 수행되고, 사용자 정의 함수는 자신이 담당하는 블록 내의 모든 정점에 접근이 가능하다. 동일한 블록 내의 정점 간의 통신은 메시지 전달 과정 없이 직접 접근이 가능하며, 메시지 전달은 다른 블록에 속한 정점과에 통신 시에만 수행된다.

블록 내의 정점들은 블록 내에 속한 정점들이 메시지 전달 과정 없이 바로 접근이 가능한 구조로 인해 추가적인 슈퍼스텝의 소요 없이 하나의 슈퍼스텝 내에 접근이 가능하다. 따라서 블록 중심 계산 모델에서는 메시지 전달 과정을 수행하기 위해 발생할 수 있는 오버헤드와 그래프 계산 작업을 위한 전체 슈퍼스텝의 수가 감소하여 전반적인 그래프 계산 작업 속도의 향상이 가능하다.

3. 관련 연구

정점 중심 그래프 처리 시스템 혹은 블록 중심 그래프 처리 시스템에서 그래프 계산 작업 수행 도중 각 계산노드의 처리 시간과 네트워크 오버헤드가 상이한 경우, 부하 불균형으로 인해 전체적인 계산 작업 처리 시간이 길어지고 성능이 저하될 수 있다. 이와 같은 작업량의 불균형을 개선시키기 위해 정점 중심 그래프 처리 시스템을 중심으로 여러 기존 연구들이 진행되어 왔다.

대부분의 기존 연구들은 그래프 계산 작업 수행 도중 계산노드 간에 걸쳐진 간선의 수를 최소화 시키는 방향으로 정점들을 동적으로 재배치하여 전반적인 네트워크 오버헤드를 줄이는 방식을 취한다. 이러한 기존 연구들은 재배치할 정점을 선택하는 방식을 기준으로 크게 모니터링 기반의 동적 정점 재배치 기법, 토폴로지(topology) 기반의 동적 정점 재배치 기법, 예측 기반의 동적 정점 재배치 기법으로 분류될 수 있다. 그러나 정점 중심 계산모델을 사용하여 정점이 개별적으로 재배치되는 방식으로 인해 블록 중심 시스템에 바로 적용이 어려우며, 블록의 연결성을 고려하지 않아 단순히 계산모델을 수정하는 것만으로는 적용에 한계가 있다.

3.1 모니터링 기반의 동적 정점 재배치 기법

모니터링 기반의 동적 정점 재배치 기법들은 실행 특성과 관련된 수치를 지속적으로 모니터링 하여 이전 슈퍼스텝의 작업 수행시간 동안 측정된 수치를 기준으로 다음 슈퍼스텝이 시작되기 전 정점을 재배치하는 방식을 취한다. GPS[9], Mizan[10], Planar[21], GrapH[22], GraphSteal[23] 등의 대부분의 기존 기법들은 이러한 모니터링 기반의 동적 정점 재배치 기법에 속하며 각 기법에 따라 측정하는 수치, 재배치할 정점을 선택하는 기준, 정점을 재배치하는 방식 등에서 조금씩 차이를 보인다.

GPS[9]에서는 정점 중심 그래프 처리 시스템에서 각 정점이 외부로 보내는 네트워크 메시지의 수를 모니터링 하여 이를 기준으로 동적으로 정점들을 재배치하는 기법을 제안하였다. GPS에서는 기본적으로 외부로 보낸 메시지의 수를 기준으로 재배치할 정점이 선정되며, 기기 당 할당된 정점의 수의 균형을 맞추기 위해 계산노드들이 서로 동일한 수의 정점을 교환하는 방식으로 정점들이 재배치된다.

Mizan[10]에서는 정점 중심 그래프 처리 시스템에서 각 정점에 대해 정점이 사용자 정의 함수를 수행한 시간, 정점이 외부로 보내는 네트워크 메시지의 수를 모니터링 하여 이를 기반으로 부하 불균형 여부를 판단하고, 부하 불균형이라고 판단되는 경우에 한해서 동적으로 정점을 재배치하는 기법을 제안하였다. Mizan의 제안 기법은 크게 실행시간 모니터링과 정점 이동 계획의 두 부분으로 나눠진다. Mizan에서는 그래프 계산 실행시간에 각 정점이 다른 정점으로 보내는 메시지의 수, 다른 정점들이 각 정점으로 보낸 메시지의 수, 각 정점이 하나의 슈퍼스텝 동안 사용자 정의 함수를 수행하는데 소요된 시간을 모니터링 하여 이를 기준으로 z-score를 계산하여 이상치가 있는지 확인 후 부하 불균형여부를 판단한다. 이러한 통계적인 계산결과 부하 불균형 상태로 판단되는 경우 정점 이동 계획이 수행된다. 정점 이동 계획이 시작될 때에는 기존에 측정된 수치들을 기반으로 네트워크 메시지와 실행시간과 관련된 상관계수를 계산하여 둘 중 더 높은 값을 가지는 항목이 최소화 목표로 설정된다. 최소화 목표가 설정되면 목표 값을 기준으로 부하가 가장 큰 계산노드와 가장 낮은 계산노드의 쌍을 생성하고, 각 쌍의 계산노드들은 서로 동일한 수의 정점을 교환하여 정점 재배치가 수행된다. 이러한 과정은 하나의 슈퍼스텝이 끝나 동기화 장벽 도달 후 다음 슈퍼스텝의 시작 전까지 수행된다.

Planar[21]는 정점 중심 그래프 처리 시스템에서 물리적 코어의 자원 경쟁(resource contention)을 고려한 정점 재배치 기법을 제안하였다. Planar에서는 Mizan과 유사한 방식으로 부하 불균형여부를 판단하여 매 반복단계마다 점진적으로 동적으로 정점 재배치 알고리즘을 수행하고, 일정 이상의 개선이 없는 경우 동적 정점 재배치 알고리즘 수행을 중단한다. Planar의 정점 재배치 알고리즘은 논리적 이동 단계와 물리적 이동 단계로 구성된다. 논리적 정점 이동 단계에서는 각 계산노드 내의 정점들을 다른 계산노드로 이동시킬 경우에 발생 가능한 네트워크 비용과 현재의 네트워크 비용 및 재배치로 인해 발생할 수 있는 네트워크 비용의 합을 비교한다. 이때 산정되는 네트워크 비용은 다른 계산노드에 대해 발생하는 외부 네트워크 비용과 하나의 계산노드 내의 코어들 간의 자원 경쟁으로 인해 발생하는 내부 네트워크 비용을 모두 고려하여 산정된다. 비교 결과 이동으로 인한 네트워크 비용 감소가 있는 경우, 감소폭이 가장 큰 계산노드를 대상으로 해당 정점이 재배치 후보로 선정된다. 물리적 정점 이동 단계에서는 재배치 후보 정점들 중 이동 대상 계산노드의 메모리 수용량을 초과하지 않는 선에서 정점 재배치가 수행된다.

GrapH[22]와 GraphSteal[23]은 정점 중심 그래프 처리 시스템들 중 정점 절단 그래프 분할 기반의 그래프 처리 시스템을 대상으로 하여 점진적으로 간선을 재배치하는 그래프 재분할 기법을 제안하였다. GrapH에서는 GPS와 유사한 방식으로 전반적인 네트워크 비용의 감소를 위해 각 계산노드가 다른 계산노드로 간선을 재배치하는 과정을 반복적으로 수행한다. GPS, Mizan, Planar는 정점의 수의 균형을 맞추기 위해 계산노드들이 동일한 수의 정점을 교환하거나 수용량을 고려하여 이동하는 정점의 수에 제한을 둔다. 반면 GrapH에서는 정점의 수에 대한 제한 없이 작업량만을 기준으로 하여 간선들을 재배치시킨다. GrapH의 각 계산노드는 GPS와 유사한 방식으로 다른 계산노드에 대해 네트워크 통신량을 특정 기준 이상

발생시키는 간선들에 대해 후보 집합을 생성한다. 그러나 두 계산노드가 동일한 수의 간선을 교환하는 대신, 두 계산노드의 작업량의 차이에 비례하는 수만큼의 간선이 이동된다. 따라서 GrapH의 경우 알고리즘 수행을 거치면서 각 분할 그래프에 속한 간선들의 수에 차이가 발생할 수 있다.

GraphSteal[23]은 Mizan[10]과 유사하게 실행 시간 모니터링 된 수치를 기반으로 동적으로 간선을 재배치 기법을 제안하였다. GraphSteal은 각 반복단계마다 실행 시간, 간선의 수, 각 계산노드의 메모리 사용/가용량을 측정하여 이를 기반으로 부하 불균형 상태를 판단하고, 부하 불균형이라고 판단될 경우 제안한 간선 재배치 알고리즘을 수행한다. 통계적인 z-core를 계산하여 이상치를 확인하고 상관계수를 계산하여 부하 불균형의 원인을 파악하는 Mizan과는 달리, GraphSteal은 통계적 계산으로 인한 부하를 줄이기 위해 간단한 휴리스틱을 기반으로 부하 불균형을 판단하고 부하 불균형일 경우 계산노드 간에 걸쳐진 간선의 수를 기준으로 간선을 재배치한다. 부하 불균형을 판단하는 휴리스틱은 실행 시간의 평균을 기반으로 계산노드들을 빠른 노드와 느린 노드로 분류하고, 느린 노드에 속하는 계산노드 중 가장 느린 노드의 실행시간이 빠른 노드로 분류되는 계산노드들의 실행시간 평균보다 일정 비율 이상 느린 경우 부하 불균형이라고 판단한다. 부하 불균형이 판단되는 경우 간선 재배치 알고리즘이 수행된다. 간선 재배치 알고리즘에서는 각 계산노드가 다른 계산노드로 보내는 네트워크 메시지의 수를 기준으로 재배치 후보 간선과 간선을 보낼 계산노드를 결정한다. 재배치 후보 간선이 선정되면 이동하고자 하는 계산노드에 대해 실행시간과 메모리 수용량을 기준으로 실제 이동 여부와 이동되는 간선의 수가 결정되어 재배치가 수행된다.

3.2 토폴로지 기반 동적 정점 재배치 기법

xDGP[24]는 정점 중심 그래프 처리 시스템에서 그래프 토폴로지만을 고려한 레이블 전파(label propagation) 방식의 동적 정점 재배치 기법을 제안하였다. xDGP는 앞서 설명한 GPS, Mizan, Planar 등 모니터링을 기반으로 실행시간, 네트워크 메시지 수 등의 실행시간 특성을 고려하여 동적으로 정점을 재배치하는 기법과는 달리, 그래프 자체의 토폴로지만을 고려하여 동적으로 정점을 재배치한다. xDGP의 동적 정점 재배치 알고리즘에서는 각 정점이 자신의 이웃들 중 가장 많은 이웃이 속한 계산노드로 이동하며, 이러한 과정은 특별한 조건 없이 모든 슈퍼스텝에 걸쳐 반복적으로 수행된다. 각 정점은 자신의 이웃 정점들이 가장 많이 속한 계산노드가 자신이 현재 속한 계산노드가 아닐 경우 이동을 결정하며, 이동할 대상의 수용량이 남는 경우 실제 이동을 수행한다. 이때 서로 연결된 일부 정점들이 반복적으로 서로의 계산노드로 재배치되는 현상이 발생할 수 있다. xDGP에서는 이러한 현상을 방지하기 위해 이동 결정시 무작위 계수를 두어 확률적으로 이동이 수행되도록 하였다.

3.3 예측 모델 기반 동적 정점 재배치 기법

LogGP는 정점 중심 그래프 처리 시스템에서 다음 슈퍼스텝에 발생할 작업 처리 시간을 예측하고 이를 기반으로 동적으로 정점을 재배치하는 기법을 제안하였다. 대부분의 모니터링 기반의 동적 정점 재배치 기법들의 경우, 이전 슈퍼스텝에 대해 모니터링 된 네트워크 메시지의 수를 기준으로 계산노드 간에 걸쳐지는 간선의 수를 최소화하는 방향으로 정점들이 재배치된다. 그러나 LogGP에서는 이전 슈퍼스텝에 대해 각 계산노드가 작업을 수행한 시간을 기록하여 이를 기준으로 다음 슈퍼스텝에 발생할 작업 처리 시간을 예측하고, 예측된 처리 시간을 기반으로 계산노드 간의 처리 시간을 고르게 맞추는 방향으로 정점들이 재배치된다.

4. 동적 블록 재배치를 통한 부하 분산 기법

본 논문에서는 블록 중심 그래프 처리 시스템에서 동적으로 블록을 재배치하여 블록의 연결성을 유지할 수 있는 로드 밸런싱 기법을 제안한다. 제안된 기법은 기존의 기법들과 동일하게 계산노드 간에 걸쳐진 간선의 수를 줄여 그래프 처리 실행시간을 줄이는 것을 목표로 한다. 그러나 기존의 기법들과는 달리 블록의 연결성을 유지하기 위해 정점이 아닌 블록 단위로 재배치를 수행한다.

일반적으로 계산노드 간의 절단 간선이 최소가 되도록 하는 문제는 NP-Complete이이며, 더 높은 정확도를 위해서 그래프 전체의 정보를 고려해야 한다. 그러나 그래프 전체에 대한 정보를 기반으로 최적의 해를 구하는 경우 해를 구하기까지 많은 시간이 소요되기 때문에 실시간으로 수행되기 어렵다. 본 논문에서 제안하는 기법은 각 계산노드 내의 블록들이 지역적 정보를 기반으로 여러 슈퍼스텝에 걸쳐 점진적으로 해를 개선한다.

본 장에서는 이러한 동적 블록 재배치 기법에 대해 자세히 설명한다. 먼저 부하 불균형을 판별하기 위해 블록 중심 그래프 처리 시스템의 작업 성능에 영향을 미치는 요인을 분석하고, 이를 기반으로 정량화된 작업 성능 모델을 설명한다. 또한 블록 재배치 알고리즘이 수행되는 전체 처리 구조와 부하 불균형 여부를 판단하는 방식, 블록 재배치를 통한 개선 정도를 판단하는 방법에 관해서 설명한다. 마지막으로는 재배치할 블록의 선정 기준, 블록이 재배치되는 조건, 블록 분할 등 블록 재배치 알고리즘의 세부사항을 설명한다.

4.1 작업 성능 모델링

블록 중심 그래프 처리 시스템에서 실시간으로 작업 부하의 불균형을 판단하기 위해서는 계산노드들 간의 작업 성능 차이를 분석하는 것이 필요하다. 이러한 작업 성능을 측정하고 분석하기 위해서는 먼저 블록 중심 그래프 처리 시스템의 작업 성능을 모델링해야 한다. 따라서 본 논문에서는 블록 중심 그래프 처리 시스템에서 작업 성능에 영향을 미치는 요인을 분석하고, 분석된 요인을 기반으로 작업 성능을 모델링하였다.

블록 중심 그래프 처리 시스템의 작업은 여러 슈퍼스텝들의 연속으로 구성된다. 하나의 슈퍼스텝에 대해, 각 계산노드는 자신이 가진 데이터를 기반으로 내부 계산 작업을 수행한

다. 작업 수행 도중 다른 계산노드에 보낼 메시지가 발생하면 해당 메시지를 내부 메시지 큐에 저장하고, 내부 계산 작업이 완료되면 내부 메시지 큐의 메시지들을 한꺼번에 전송한다. 따라서 하나의 계산노드가 하나의 슈퍼스텝을 완료하는 데에 걸리는 시간은 내부 계산 작업을 수행한 시간과 메시지 전송을 위한 통신 시간으로 구성된다. 이때 내부 계산 작업 처리 시간과 네트워크 메시지의 수 및 전송시간은 계산노드에 따라 다를 수 있다. 결과적으로 전체 처리 시스템이 하나의 슈퍼스텝을 완료하는 데에 걸리는 시간은 가장 느린 계산노드에 의해 결정된다. Fig. 4는 이러한 슈퍼스텝 내 계산노드의 작업 성능 구성요소를 나타낸다.

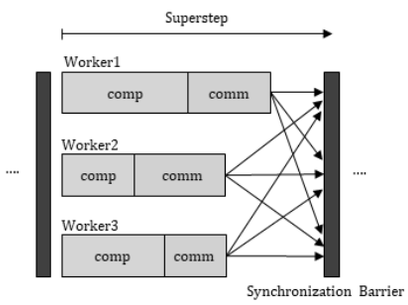

Fig. 4. Performance Factor of Node Per Superstep

Bt_{comp}를 하나의 블록이 하나의 슈퍼스텝동안 사용자 정의 함수를 처리하는 시간이라고 했을 때, n개의 블록을 가진 계산노드가 하나의 슈퍼스텝동안 작업을 수행하는 시간 Wt_{comp}은 Equation (1)과 같이 노드 내의 블록들 중 가장 처리가 느린 블록의 사용자 정의 함수 처리 시간으로 정의될 수 있다. 또한 Bn_{msg}이 하나의 블록이 하나의 슈퍼스텝동안 사용자 정의 함수를 처리하면서 다른 계산노드에 속한 블록으로 보내는 메시지의 수라고 했을 때, n개의 블록을 가진 계산노드가 하나의 슈퍼스텝동안 다른 계산노드로 보내는 메시지의 수 Wn_{msg}는 Equation (2)와 같이 모든 블록들의 Bn_{msg}의 합으로 정의될 수 있다. 따라서 하나의 계산노드가 하나의 슈퍼스텝동안 계산 작업과 통신 작업을 수행하는 전체 시간 W_{time}은 Equation (3)과 같이 정의할 수 있다. 여기서 α는 네트워크 메시지를 전송하는데 걸리는 시간에 대한 상수를 의미한다.

$$Wt_{comp} = \max(Bt^{1}_{comp}, \ldots, Bt^{n}_{comp}) \quad (1)$$

$$Wn_{msg} = \sum_{k=1}^{n}(Bn^{k}_{msg}) \quad (2)$$

$$W_{time} = Wt_{comp} + \alpha * Wn_{msg} \quad (3)$$

4.2 블록 재배치 수행 단계 및 처리 구조

본 논문에서 제안하는 기법은 각 계산노드 내의 블록들이 지역적인 정보만을 가지고 여러 슈퍼스텝에 걸쳐 점진적으로 해를 개선해나가는 방식을 취한다. 제안 기법은 블록 중심 그래프 처리 시스템의 작업 성능 모니터링을 통해 성능의 불균형이 발생하면 동적 블록 재배치 모드를 활성화 한다. 동적 블록 재배치 모드가 활성화 되면, Fig. 5와 같이 하나의 슈퍼스텝에서 계산노드가 작업을 마치고 슈퍼스텝 동기화 장벽에 도달하여 블록 재배치 과정이 수행되고 그 이후 다음 슈퍼스텝이 시작된다. 이러한 실행 흐름은 동적 블록 재배치 모드가 비활성화 될 때까지 각 슈퍼스텝의 끝에 수행된다.

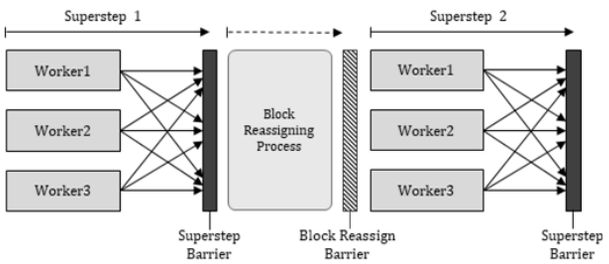

Fig. 5. Flow of Block Reassigning Process

1) Balancing Manager

Balancing Manager는 Master 노드 내에 상주하는 프로세스로, 블록 재배치 과정 전체의 실행 흐름을 제어하는 역할을 수행한다. Balancing Manager는 각 슈퍼스텝의 끝에 계산노드들의 W_{time}을 전달받아 이를 기반으로 작업 성능을 모니터링 한다. 모니터링 결과 해당 슈퍼스텝의 작업 성능에 부하 불균형이 발생했다고 판단되면, Balancing Manager는 동적 블록 재배치 모드를 활성화시켜 동적 블록 재배치 알고리즘을 수행한다. 동적 블록 재배치 모드가 활성화되면 모든 슈퍼스텝의 끝에 블록 재배치 과정이 수행된다. Balancing Manager는 여러 슈퍼스텝동안 블록 재배치 수행으로 인한 개선 정도를 분석하여 더 이상의 개선이 없다고 판단되면 동적 블록 재배치 모드를 비활성화 시켜 블록 재배치 과정의 수행을 종료한다. Fig. 6은 이러한 Balancing Manager가 하나의 슈퍼스텝을 완료한 후 수행하는 전체 처리과정을 순서도로 나타낸 것이다.

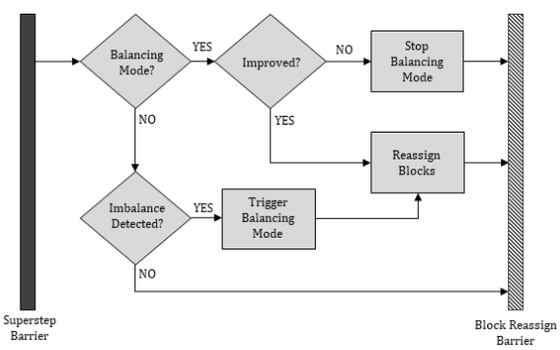

Fig. 6. Process Step of Balancing Manager Per Superstep

2) 부하 불균형 여부 판단

Balancing Manager는 모니터링 된 계산노드들의 W_{time} 값들 중 이상치가 있는지를 분석하여 부하 불균형 여부를 판단

한다. Mizan의 경우 모니터링 되는 모든 수치들에 대해 표준 점수를 계산하고, 이를 기준으로 분포에서 벗어난 이상치가 있는지를 분석하여 부하 불균형 여부를 판단한다[10]. 그러나 이러한 통계적인 계산이 모든 슈퍼스텝의 끝에 수행되는 경우, 계산에 대한 오버헤드로 인해 실시간 처리 성능이 저하될 수 있다. GraphSteal에서는 이러한 실시간 통계 계산과 관련된 오버헤드를 줄이기 위해 모니터링 측정치의 최대, 최소의 값만 고려하는 간단한 휴리스틱을 제안하고, 이러한 휴리스틱 기법이 실제 부하 불균형 판별에 효과가 있음을 입증하였다[23]. 본 논문에서는 이러한 GraphSteal의 제안 기법을 사용하여 부하 불균형 여부를 판단한다.

3) 블록 재배치 과정 종료

본 논문에서 제안하는 동적 블록 재배치 기법은 계산노드 간의 절단 간선 수를 줄이도록 재배치된다. 그러나 일정 시점에 도달하면 동적 블록 재배치가 수행되어도 절단 간선 수가 더 이상 줄어들지 않고 일정하게 유지될 수 있다. 이 경우에는 동적 블록 재배치로 인한 성능의 개선보다 블록 재배치 수행 과정으로 인한 오버헤드가 더 커질 수 있다. 따라서 본 논문의 제안 기법은 성능의 개선이 더 이상 없다고 판단되면 블록 재배치를 수행하지 않는다.

Balancing Manager는 불필요한 블록 재배치 과정을 피하기 위해 계산노드 간의 절단 간선 수의 개선 정도를 분석하고, 그 결과에 따라 동적 블록 재배치 과정 수행 여부를 결정한다. 그러나 동적 블록 재배치 과정은 각 블록에 대한 지역적인 정보만을 가지고 수행되기 때문에 수행 초기에는 개선의 정도가 나쁠 수 있다. 만일 개선의 정도를 판단하기 위해 바로 이전 단계와 현재 단계를 비교하게 되면, 장기적으로 블록 재배치 과정을 수행하여 성능의 개선을 얻을 수 있음에도 불구하고 블록 재배치 수행이 초기에 종료되어 지역 최적해에 머무를 수 있다. 따라서 Balancing Manager는 일정 주기 δ를 기준으로 절단 간선 $edgecut$의 개선 정도를 고려한다. Balancing Manager는 임의의 슈퍼스텝 s에서 개선 정도 $Rate_{improved}$를 Equation (4)와 같이 계산한다. 만일 $Rate_{improved}$가 1보다 큰 값을 가지는 경우, 개선이 이루어지지 않았다고 판단하고 블록 재배치 과정을 종료한다.

$$Rate_{improved} = \frac{edgecut_s}{edgecut_{s-\delta}} \quad (4)$$

4.3 블록 재배치 알고리즘

본 논문에서 제안하는 기법은 블록의 연결성을 유지하기 위해 블록 단위로 재배치를 수행하여 절단 간선의 수를 점진적으로 줄여나간다. 제안 기법의 블록 재배치 알고리즘은 각 블록을 블록과 연결된 간선이 가장 많은 계산노드로 재배치하여 절단 간선의 수를 줄이고, 필요에 따라 블록의 일부분만 분할하여 재배치한다. 본 절에서는 이러한 블록 재배치 알고리즘에서 재배치될 블록이 선정되는 기준과 실제로 블록이 재배치되는 조건에 대해 자세히 다룬다. 또한 블록 분할이 필요한 상황과 이에 따른 블록 분할과정에 대해서 설명한다.

1) 블록 재배치 알고리즘

블록 재배치 알고리즘은 크게 블록 재배치 대상 선정, 블록 재배치 요청 전송, 블록 재배치 요청 처리, 블록 재배치 요청 결과 반환, 블록의 이동, 이동된 블록의 합병과정으로 구성된다. Fig. 7은 이러한 블록 재배치 알고리즘의 전체 수행 과정을 나타낸다.

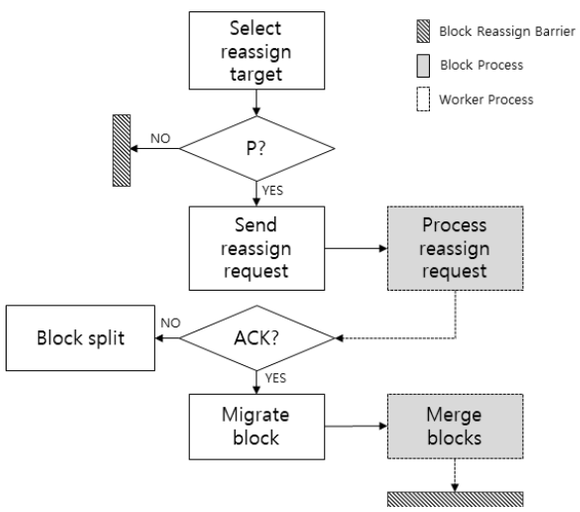

Fig. 7. Flow of Block Reassignment Process

a) 블록 재배치 대상 선정

블록이 재배치될 계산노드는 블록과 계산노드들 간의 절단 간선의 수를 기준으로 선정된다. 각 블록은 블록과 계산노드 간의 절단 간선 수를 계산하기 위해 자신이 가진 정점 중 다른 계산노드에 이웃을 가지는 정점을 찾고, 이러한 정점들이 외부와 연결된 간선 수를 계산노드 별로 집계한다.

b) 확률 p에 따른 블록 재배치 요청 전송

기본적으로 모든 블록이 모든 슈퍼스텝 마다 재배치되는 경우 다음과 같은 문제가 발생할 수 있다. 첫째로, 각 블록은 자신의 정보만 가지고 재배치될 계산노드를 선택하므로 다른 블록들이 어떤 계산노드로 재배치되는지 알 수 없다. 따라서 Fig. 8과 같이 특정 블록들이 반복적으로 교환되어 서로를 쫓는 현상이 나타날 수 있다. 둘째로, 블록의 크기가 큰 경우 반복적으로 재배치되는 것보다 블록이 속한 계산노드에 남아있는 것이 네트워크 비용 측면에서 유리하다. 본 논문에서는 이러한 문제점을 해결하기 위해 확률적으로 블록을 재배치하는 방법을 제안한다.

각 블록은 재배치 대상을 선정하고 난 후 해당 계산노드로 (블록식별자, 절단 간선 수, 블록 크기) 정보를 포함하는 재배치 요청을 전송한다. 그러나 이러한 재배치 요청 전송은 p의 확률로 발생한다. 확률 p는 Equation (5)와 같이 모든 블록의 평균 크기를 블록의 크기로 나눈 값으로 계산된다. 평균보다

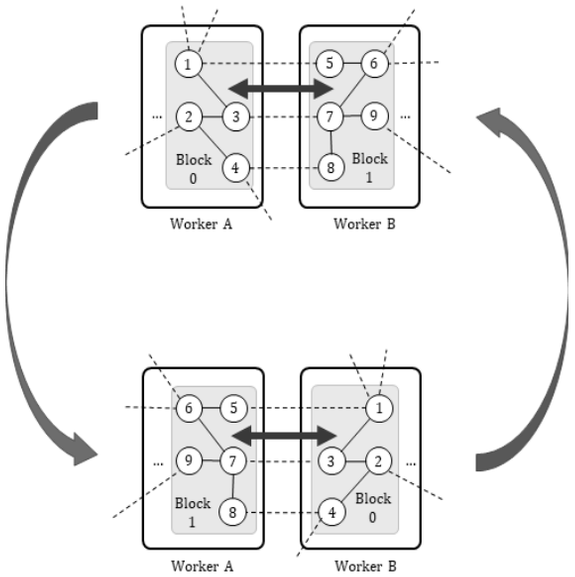

Fig. 8. Iterative Block Exchange

작은 크기를 가지는 블록의 경우 이러한 확률 p의 값이 항상 $p \geq 1$ 이 되므로 무조건 재배치 요청을 전송한다. 그러나 평균보다 큰 크기를 가지는 블록의 경우 0과 1사이의 난수를 생성하여 구해진 값이 p보다 작은 경우에만 재배치 요청을 전송한다. 이때 확률 p의 값은 블록의 크기가 클수록 작아지므로, 블록의 크기가 큰 블록일수록 낮은 확률로 재배치 요청을 전송하게 된다.

$$p = \frac{avg_{size}}{B_{size}} \tag{5}$$

c) 블록 재배치 요청 처리 및 수락

블록으로부터 재배치 요청을 받은 계산노드는 블록과 계산노드 간의 절단 간선 수와 블록의 크기를 고려하여 재배치 요청을 수락한다. 하나의 계산노드는 노드가 가진 블록들의 크기에 따라 새로 수용할 수 있는 블록들의 수와 크기에 제한이 있다. 또한, 계산노드와 재배치 요청을 보낸 블록 간의 절단 간선 수가 많을수록 재배치로 인한 전체 절단 간선 수의 감소폭이 더 크므로, 계산노드와 블록 간의 절단 간선 수가 더 많은 요청부터 수락하는 것이 더 유리하다. 이를 위해 각 계산노드는 자신이 받은 재배치 요청들을 절단 간선 수가 많은 순으로 정렬하고, 자신의 수용량을 넘지 않을 때까지 가장 상위의 요청부터 순차적으로 수락하는 과정을 거친다.

d) 블록 이동 및 블록 합병

재배치요청 처리가 완료되면 각 계산노드는 재배치 요청의 결과를 블록들에게 반환한다. 만일 요청이 수락되면 각 블록은 블록의 상태, 블록이 가진 정점들의 정보를 대상 계산노드로 전송한다. 데이터 이동이 완료되고 나면 각 계산노드가 재배치된 블록들의 결과를 종합하고, 재배치된 블록으로 인해 서로 연결 가능한 블록들을 하나의 큰 블록으로 합병한다. 블록 중심 그래프 처리 시스템의 블록은 연결된 부분 그래프로 정의되기 때문에 이와 같이 재배치 후 연결 가능한 블록들을 하나의 블록으로 합병하면 블록 중심 그래프 처리 시스템의 이점을 극대화할 수 있다.

2) 블록 분할 알고리즘

본 논문에서 앞서 제안한 블록 재배치 알고리즘은 정점들이 하나의 블록을 단위로 재배치되며, 크기가 큰 블록일수록 낮은 확률로 재배치 요청을 보낸다. 이러한 블록들은 재배치 요청 전송 확률 자체가 낮을 뿐만 아니라, 재배치 요청을 보내더라도 블록의 크기가 계산노드들의 수용 량을 쉽게 초과하므로 요청이 수락되기 어렵다. 이는 네트워크 비용 측면에는 유리하지만, 블록의 일부분만을 재배치하여 더 나은 상태로 가는 것을 사전에 차단하여 지역 최적해에 머물게 할 수 있다.

본 논문에서는 이러한 경우를 고려하여 블록 재배치 알고리즘(Fig. 9)이 특정 경우에 블록 분할을 수행할 수 있도록 하였다. 이를 위해 각 블록은 전송한 재배치 요청이 거절되는 경우 재배치 요청을 시도했던 횟수 R_{count}를 기록하고, 이러한 R_{count}가 특정 횟수 γ 이상 계속되면 블록 분할 알고리즘을 수행한다. 이때 수행되는 블록 분할 알고리즘은 논리적 블록 분할 단계와 블록 분할로 얻을 수 있는 이득의 판단, 물리적 블록 분할의 세 단계로 구성된다.

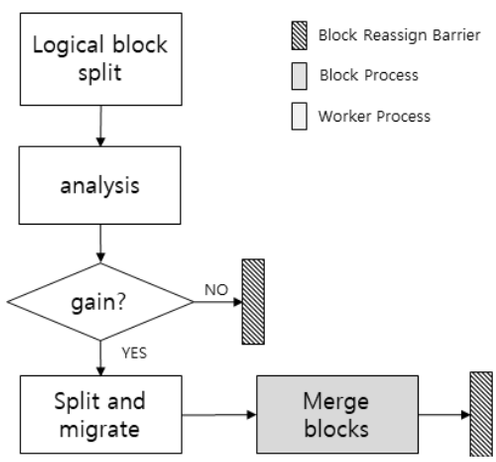

Fig. 9. Flow of Block Split Process

a) 논리적 블록 분할

블록은 가장 먼저 내부 절단 간선의 수가 최소가 되도록 k개로 분할하는 과정을 수행하며, 이러한 과정은 실제 블록 분할로 이어지지 않고 논리적으로만 수행된다. 분할되는 수 k는 블록이 다른 계산노드와 절단 간선을 가질 때, 이러한 외부 절단 간선을 가지는 계산노드들의 수로 결정된다. 그러나 하나의 블록을 최소의 절단 간선을 가지도록 k개로 분할하는 문제는 결국 그래프 분할 문제와 동일하게 NP-Complete이다. 따라서 본 논문에서는 블록을 최적으로 분할하는 문제는 다루지 않으며,

대신 기존에 제안된 레이블 전파(label propagation) 기반의 그래프 분할기법[17]을 수정하여 적용하였다.

b) 블록 분할로 인한 이득 판단

논리적 블록 분할이 완료되면 가장 많은 절단 간선 수를 가지는 부분 블록에 대해, 블록이 현재 다른 계산노드와 연결된 절단 간선 수 $edgecut_{out}$과 블록이 이동할 경우 새롭게 발생하는 절단 간선의 수 $edgecut_{inner}$를 계산한다. 만일 $edgecut_{out}$보다 $edgecut_{inner}$가 더 많다면, 부분적으로 블록을 재배치했을 때 절단 간선의 수가 오히려 더 늘어나므로 이득이 없다고 판단한다. 반대로 $edgecut_{out}$가 $edgecut_{inner}$보다 많은 경우, 부분적으로 블록을 재배치했을 때 전체 절단 간선 수가 감소하므로 이득이라고 판단한다.

c) 물리적 블록 분할

$edgecut_{out}$과 $edgecut_{inner}$를 비교했을 때 이득이 있다고 판단되는 경우에는 실제 블록 분할 과정이 수행되고, 분할된 부분의 블록이 대상 계산노드로 재배치된다. 만일 이득이 없다고 판단되는 경우에는 블록 분할 및 재배치 과정을 수행하지 않고 블록 분할 알고리즘이 종료된다.

5. 성능 평가

본 논문에서는 제안 기법의 성능평가를 위해 블록 중심 그래프 처리 시스템의 핵심 기능을 가지는 시스템을 구현하고, 구현한 시스템 상에서 제안 기법을 적용한 경우와 적용하지 않은 경우에 대한 그래프 처리 실험을 진행하였다. 기존의 기법들은 정점 중심 그래프 처리 시스템 기반으로 정점을 동적으로 배치하는 기법이기 때문에, 직접적인 비교 보다는 블록 중심 그래프 처리 시스템 기반으로 제안 기법을 적용한 방식과 적용하지 않은 경우를 비교하였다.

두 경우 모두 동일한 그래프와 사용자 정의 함수를 사용 및 초기 정점은 랜덤으로 배치하였으며 데이터의 크기에 따른 성능의 변화 경향을 살펴보기 위해 여러 데이터에 대한 실험이 진행되었다. 또한 확률 p는 Equation (5)를 사용하여 구하였으며, γ 값은 5로 설정해서 실험을 진행하였다. 설정한 γ 값이 작은 경우는 크기가 큰 블록들의 요청에 대해 분할 알고리즘 수행이 잦아지면서 추가 overhead가 발생하지만, 지역 최적해에 머물 확률은 줄어들게 된다. 반면, γ 값이 큰 경우 분할 알고리즘의 수행 빈도가 낮아져서 이에 대한 overhead의 감소를 기대할 수 있으나, 블록 재배치 요구의 수행이 늦어져서 지역 최적해에 머무는 확률이 증가되게 된다.

5.1 성능 평가 환경

1) 실험 환경

본 실험의 진행을 위해 먼저 아마존 웹 서비스(AWS)의 컴퓨팅 자원 4개에 대해 클러스터를 구성하였다. 클러스터 구성을 위해 사용된 AWS 컴퓨팅 자원의 상세한 사양은 Table 1과 같다.

Table 1. Experiment Environment

	Node1	Node2	Node3	Node4
Type	EC2-r4.xlarge			
CPU	Intel Broadwell, 2.3 GHz			
vCPUs	4			
Memory(GiB)	30.5 GiB			
Storage(GB)	Amazon Elastic Block Store(EBS), General Purpose (SSD) Volume			
OS	Ubuntu Server 16.04 LTS			
Cluster	Master	Worker	Worker	Worker

2) 실험 데이터

실험 데이터로는 KONECT[26]에서 제공하는 그래프 데이터와 정보를 사용하였으며, 데이터의 크기 및 그래프 특성에 따른 성능의 변화 경향을 살펴보기 위해 여러 데이터에 대해 실험을 진행하였다. 실험에 사용된 그래프 데이터에 대한 상세한 정보는 Table 2와 같다.

Table 2. Description of Experiment Data

	Data1	Data2	Data3	Data4	Data5
Name	Amazon (MDS)	Youtube friend ship	Skitter	Flicker links	Orkut
Code	CA	CY	SK	LF	OR
File Size (KB)	12,291	85,730	145,612	189,948	1,768,888
# of Vertex	334,863	1,134,890	1,696,415	1,715,255	3,072,441
# of Edge	925,872	2,987,624	11,095,298	15,551,250	117,184,899
Average degree	5.5299	5.265	13.081	18.133	76.281

Amazon (MDS)는 아마존에서 동일한 상품을 구매한 고객들을 기반으로 생성된 가공 그래프로, 각 상품들이 하나의 정점이 되어 구매 고객을 공유하는 상품들 간에 연결 관계가 형성된다. Youtube friendship, Flicker link, Orkut은 각 사이트의 사용자들에 대한 소셜 그래프로, 사이트의 사용자가 하나의 정점이 되어 친구 관계에 있는 사용자들이 간선으로 연결된다. Skitter는 Skitter의 트레이스 라우트(traceroute)로부터 생성된 인터넷 토폴리지 그래프로, 각 자율 시스템(Autonomous System)들이 하나의 정점이 되어 서로 연결된 시스템들에 대해 간선이 생성된다.

3) 시스템 구현

실험을 위해 구현된 시스템의 아키텍처는 Fig. 10과 같다. 구현 시스템은 하나의 Master 노드와 세 개의 Worker 노드 상에서 동작하며, UbuntuOS 상에 Java(jdk 1.8)를 기반으로 구

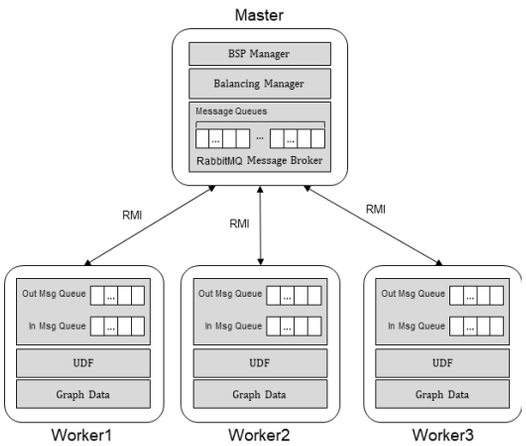

Fig. 10. System Implementation Architecture

현되었다. Master에는 BSP 처리의 실행 흐름을 담당하는 BSP Manager와 제안 기법의 실행 흐름을 제어하는 Balancing Manager가 상주하며, 블록 간의 메시지 통신을 위한 메시지 브로커(message broker)를 가진다. Worker는 처리할 그래프 데이터와 사용자 정의 함수를 가지며, 다른 노드 내의 블록에게 보낼 메시지와 자신의 블록이 받는 메시지를 임시로 저장하는 큐를 가진다. 기본적으로 BSP의 동기화를 위한 노드 간의 통신은 Java Remote Method Invocation (RMI)를 사용하였으며, 그래프 처리를 위한 블록 간의 통신은 RabbitMQ[25]를 사용하여 구현되었다.

5.2 성능평가 결과

성능 평가를 위해 구현된 시스템 상에서 제안 방식을 적용한 경우와 적용하지 않은 경우에 대한 실험이 진행되었다. 제안 방식과 기존 방식 모두 동일한 사용자 정의 함수와 그래프 데이터에 대해 4개의 노드에서 실험을 수행하였으며, 사용자 정의 함수로는 그래프 알고리즘 성능을 분석하는데 자주 사용되는 그래프 내 최솟값 찾기 알고리즘이 사용되었다.

최종적으로 수행된 실험 결과는 Table 3과 같으며, Fig. 11은 Table 3의 처리 시간에 대한 비교를 보여준다. 결과적으로 CA를 제외한 나머지 데이터에 대해 제안 방식이 평균 10.3% 정도 처리 시간의 향상이 있었으며, 데이터의 크기, 그래프의 정점 및 간선의 수, 그래프 정점의 평수 차수에 따라 처리 향상률이 조금씩 커지는 경향을 확인하였다. CA는 실험 데이터 중 가장 작은 규모의 그래프 데이터로, 실험 결과 처리 성능이 더 나쁜 결과를 보여주었다. 이는 본 논문의 제안기법이 규모가 작은 데이터에서는 오버헤드로 작용하는 것으로 보인다.

Table 3. Improvement of Execution Time

Data	NB	DB	Improvement(%)
CA	210.25	212.21	-0.92
CY	1518.06	1390.68	9.16
SK	7374.78	6728.04	9.61
LF	7731.13	7028.16	10.00
OR	67626.73	60123.25	12.48

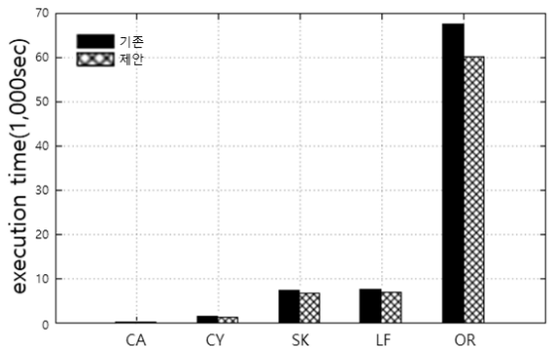

Fig. 11. Comparison of Execution Time

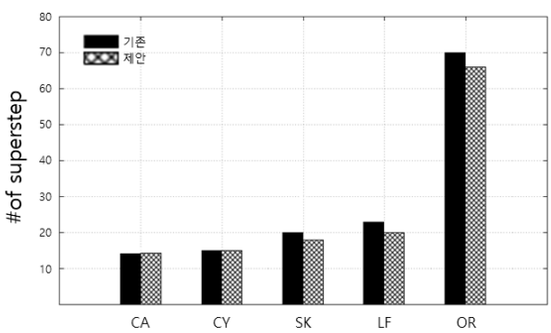

Fig. 12. Comparison of Supersteps

Fig. 12는 수행된 실험의 슈퍼스텝 수를 비교한 그래프이다. 한편 슈퍼스텝의 경우 기존 기법과 제안 기법의 차이가 크지 않거나 제안 기법이 기존 기법에 비해 슈퍼스텝이 약간 감소하였다. 이는 제안 기법이 그래프 최솟값을 찾기까지 수행되는 단계가 비슷한 수준을 유지하는 것을 의미한다. 다만 처리 시간과 마찬가지로 CA의 경우 오히려 슈퍼스텝이 소폭 증가한 것을 확인하였다. 이는 그래프의 데이터 크기가 작은 경우에 자체적으로 외부로 나가는 간선 수가 적고, 이에 따라 오히려 제안 방식인 동적으로 파티션을 적용할 때 외부로 나가는 간선 수가 조금 더 많아지는 경우가 발생하기 때문이라고 추측한다.

6. 결론 및 향후 연구

본 논문에서는 블록 중심 그래프 처리 시스템의 로드 밸런싱을 위한 동적 블록 재배치 기법을 제안하였다. 본 논문의 제안 기법은 블록의 지역적인 정보만을 가지고 점진적으로 블록 재배치를 동적으로 수행하여 분할 그래프 간 절단 간선 수를 감소시켜 전체적인 처리 성능의 향상을 얻고자 하였다. 실험 결과 대부분의 데이터에 대해 제안기법이 기존 기법에 비해 비슷한 수준의 슈퍼스텝 수를 유지하면서도 평균 10.3%의 처리 시간의 향상이 있음을 확인하였다.

제안 기법의 처리 성능은 그래프 데이터의 규모가 커질수록 향상되는 정도가 함께 커졌으나 그래프 데이터의 특성으로 인해 정확한 상관관계를 찾기 어렵다. 또한 제안 기법 내에 확률적 요소가 존재하므로, 실험 결과를 통해 나온 결과가 제안

기법의 효과를 완벽히 설명한다고 볼 수 없다. 따라서 향후에는 그래프 데이터 내의 연결 특성과 확률적 요소의 변화에 따른 제안 기법의 효과를 보다 자세히 분석하는 연구를 진행하고자 한다. 또한, 제안 논문에서는 4개의 클러스터를 통해 실험을 진행 하였으며, 클러스터의 수를 증가함에 따른 추가 실험을 통해 확장성을 보여주는 연구를 진행하고자 한다.

References

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I., Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," In *Proc. 2010 ACM SIGMOD International Conference on Management of Data,* ACM, pp.135-146, 2010.

[2] The Apache Software Foundation, "Welcome to ApacheTM Hadoop®!," The Apache Software Foundation, 2014. [Online]. Available: http://hadoop.apache.org. [Accessed Dec. 1, 2017].

[3] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," In *Proc. IEEE 9th International Conference on Data Mining*, IEEE, pp. 229-238, 2009.

[4] J. Lin and M. Schatz, "Design patterns for efficient graph algorithms in MapReduce," In *Proc. 8th Workshop on Mining and Learning with Graphs,* ACM, pp.78-85, 2010.

[5] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, Vol.51, No.1, pp.107-113, 2008.

[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," In *Proc. VLDB Endowment*, Vol.5, No.8, pp.716-727, 2012.

[7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," *OSDI*, Vol.12, No.1, p.2, 2012.

[8] J. E. Gonzalez, R. S., Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework," *OSDI*, Vol.14, pp.599-613, 2014.

[9] S. Salihoglu and J. Widom, "Gps: A graph processing system," In *Proc. 25th International Conference on Scientific and Statistical Database Management*, ACM, 2013, p.22.

[10] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, "Mizan: a system for dynamic load balancing in large-scale graph processing," In *Proc. ACM 8th European Conference on Computer Systems*, ACM, 2013, pp. 169-182.

[11] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, Vol.33, No.8, pp.103-111, 1990.

[12] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From think like a vertex to think like a graph," In *Proc. VLDB Endowment*, Vol.7, No.3, pp.193-204, 2013.

[13] Y. Simmhan, A. Kumbhare, C. Wickramaarachchi, S. Nagarkar, S., Ravi, C., Raghavendra, and V. Prasanna, "Goffish: A sub-graph centric framework for large-scale graph analytics," In *Proc. 20th European Conference on Parallel Processing,* Springer, Cham, pp. 451-462, 2014.

[14] D. Yan, J., Cheng, Y. Lu, and W. Ng, "Blogel: A block-centric framework for distributed computation on real-world graphs," In *Proc. VLDB Endowment*, Vol.7, No.14, pp.1981-1992, 2014.

[15] S. Aridhi, A. Montresor, and Y. Velegrakis, "BLADYG: A novel block-centric framework for the analysis of large dynamic graphs," In *Proc. ACM Workshop on High Performance Graph Processing*, ACM, pp. 39-42, 2016.

[16] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, Vol.1, No.3, pp. 237-267, 1976.

[17] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, Vol.20, No.1, pp.359-392, 1998.

[18] P. Sanders and C. Schulz, "Engineering Multilevel Graph Partitioning Algorithms," *ESA*, Vol.6942, pp.469-480, 2011.

[19] A. J. Soper, C. Walshaw, and M. Cross, "A combined evolutionary search and multilevel optimisation approach to graph-partitioning," *Journal of Global Optimization*, Vol.29, No.2, pp.225-241, 2004.

[20] N. Xu, L. Chen, and B. Cui, "LogGP: a log-based dynamic graph partitioning method," In *Proc. VLDB Endowment*, Vol.7, No.14, pp. 1917-1928, 2014.

[21] A. Zheng, A. Labrinidis, and P. K. Chrysanthis, "Planar: Parallel lightweight architecture-aware adaptive graph repartitioning," In *Proc. IEEE 32nd International Conference on Data Engineering*, IEEE, pp.121-132, 2016.

[22] C. Mayer, M. A. Tariq, C. Li, and K. Rothermel, "Graph: Heterogeneity-aware graph computation with adaptive partitioning," In *Proc. IEEE 36th International Conference on Distributed Computing Systems*, IEEE, pp.118-128, 2016.

[23] D. Kumar, A. Raj, and J. Dharanipragada, "GraphSteal: Dynamic Re-Partitioning for Efficient Graph Processing in Heterogeneous Clusters," In *Proc. IEEE 10th International Conference on Cloud Computing*, IEEE, pp.439-446, 2017.

[24] L. M. Vaquero, F. Cuadrado, D. Logothetis, and C. Martella, "Adaptive partitioning for large-scale dynamic graphs," In *Proc. IEEE 34th International Conference on Distributed Computing Systems*, IEEE, pp. 114-153, 2014.

[25] Pivotal Software, "RabbitMQ-Messaging that just works," Pivotal Software, 2007. [Online]. Available: https://www.rabbitmq.com. [Accessed Dec. 1, 2017].

[26] J. Kunegis, "KONECT - The Koblenz Network Collection," uni-koblenz.de, Apr. 25, 2017. [Online]. Available: http://konect.uni-koblenz.de. [Accessed Dec. 7, 2017]

김 예 원

https://orcid.org/0000-0002-4309-9923
e-mail : kimyewon@ajou.ac.kr
2016년 아주대학교 미디어학과(학사)
2018년 아주대학교 컴퓨터공학과(석사)
2018년～현 재 리디북스 사원
관심분야 : 병렬/분산 처리, 그래프 처리

배 민 호

https://orcid.org/0000-0003-3342-1779
e-mail : minkkang@ajou.ac.kr
2012년 아주대학교 정보컴퓨터공학부(학사)
2012년～현 재 아주대학교 컴퓨터공학과 석・박통합과정
관심분야 : 병렬/분산 처리, 그래프 처리, 시멘틱 웹

오 상 윤

https://orcid.org/0000-0001-5854-149X
e-mail : syoh@ajou.ac.kr
2006년 미국 인디애나대학교 컴퓨터공학과 (박사)
2006년～2007년 SK텔레콤 전략기술부문
2007년～현 재 아주대학교 소프트웨어학과 교수
관심분야 : 분산/병렬 시스템, 고성능컴퓨팅, 클라우드컴퓨팅, Semantic Web