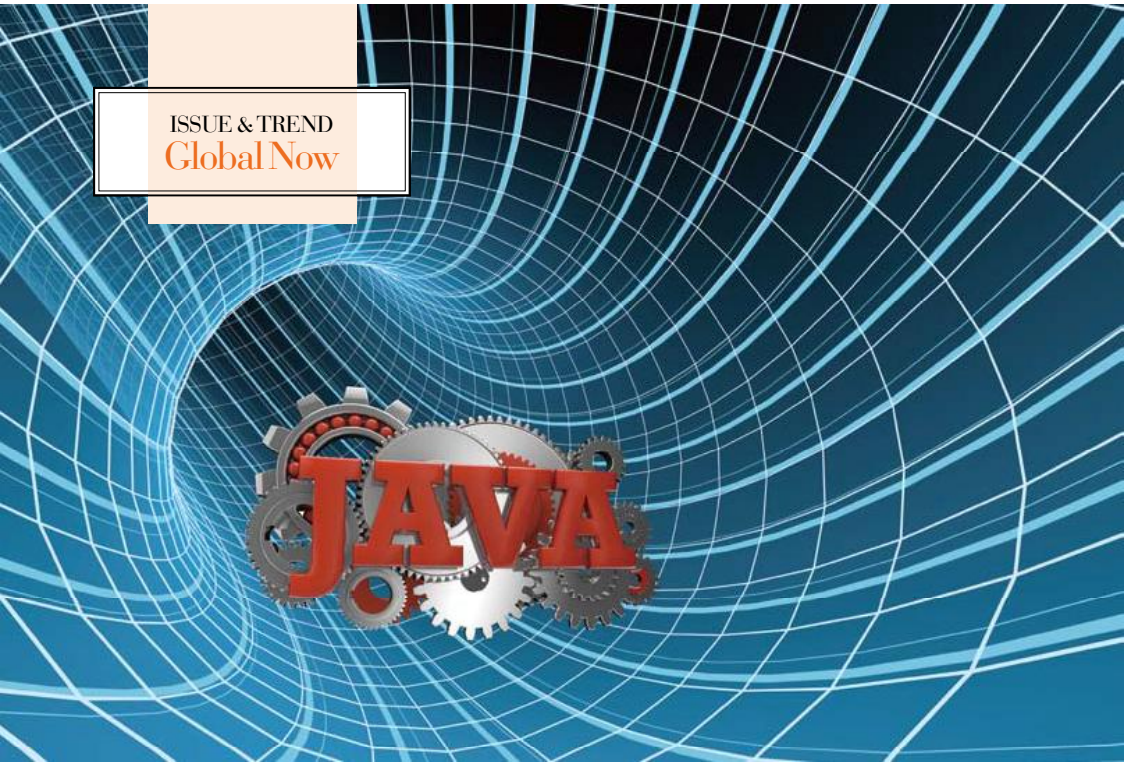


# JAVA로도 고속 트레이딩이 가능하다

**EXTURE⁺의 등장으로 거래 속도가 획기적으로 빨라질 전망이다. 하지만 이를 위한 시스템은 천편일률적이다. 최근 미국에서 이런 상황을 깨고 자바를 이용한 고속 트레이딩 시스템들이 속속 등장하고 있다.**

**글** 김형준(트레이딩컨설팅그룹이음 대표파트너) **사진** 루키야노바 나탈리아, 프렌타

**EXTURE⁺란**

한국거래소와 코스콤(Koscom)이 개발 중인 차세대 시스템 EXTURE⁺는 리눅스 운영체제와 속도가 개선된 x86서버, 그리고 디스크 액세스 기반 DB에서 메모리 액세스 DB로의 전환 등을 통해 기존의 EXTURE보다 약 280배 더 빠른 속도를 자랑한다.

2009년 오라클에 인수된 썬마이크로시스템즈가 만들어서 공개한 자바(JAVA). 객체 지향 프로그래밍 언어인 JAVA는 보안성이 뛰어나며 컴파일한 코드는 다른 운영체제에서 사용할 수 있도록 클래스(class)로 제공되는 특징이 있다. JAVA의 편리성을 바탕으로 다양한 프로그램들이 제작되었고 과거 금융권에서도 JAVA를 이용한 프로그램 개발이 왕성했었다. 그러나 1990년대 말 JAVA 스윙(JAVA Swing)을 이용한 HTS 개발이 실패한 이후 JAVA는 자본시장에서 웹시스템의 개발 언어로만 자리매김하게 된다. 그러다 2009년에 대신증권이 JAVA를 이용해 원장시스템을 개발하면서 JAVA는 계정계 업무시스템까지 적용 범위를 넓혀가기 시작했다. 그리고 미국에서는 JAVA를 이용해 로우 레이턴시(Low Latency)에 맞는 매매 프로그램이 속속 등장하고 있다.

## 트레이딩의 변화

2014년 1분기 중에 EXTURE⁺가 선을 보일 예정이다. 막대한 금액을 들여 EXTURE에서 EXTURE⁺를 도입하게 한 가장 큰 원동력은 머신 트레이딩(Machine Trading, 시스템 트레이딩에 머신이 스스로 학습하여 최적화한 방법을 찾는 것까지 포함하는 신용어)의 등장이다. 수학적인 알고리즘으로 무장한 컴퓨터가 자동으로 실행하는 매매전략이 시장의 주류로 자리 잡았기 때문이다.

미국과 유럽에서는 2005년부터 머신 트레이딩 비중이 증가해 2010년을 전후한 시점에는 이를 통한 주문이 60~70%에 달했다.

국내의 경우 공식적인 통계는 없다. 하지만 지난 2009년 주문 전송망 논란은 머신 트레이딩이 한국 자본시장에서 주요 흐름을 형성하고 있음을 보여주는 상징적인 사건이었다. 이후 ELW(주식워런트증권)사건을 거치면서 주문수탁제도가 변경돼 DMA(직접주문전용선)서비스가 공식화됐다.

머신 트레이딩의 특징은 두 가지다. 하이 프리퀀시(High Frequency)와 로우 레이턴시다. 이 두 가지를 이용해 사람의 개입 없이 짧은 시간에 대량 주문이 동시 다발적으로 나온다. 매매체결시스템이 이런 머신 트레이딩에 대응하려면 초당 처리 건수와 건당 응답 속도 모두 개선돼야 한다. 99.999%의 신뢰성을 특징으로 하는 EXTURE가 신뢰성을 유지하면서 성능을 높인 EXTURE⁺로 바뀌어야 하는 까닭이다. EXTURE⁺가 세운 목표는 속도의 경우 70마이크로세컨드, 스루풋(Throughput)은 20,000 TPS다. 즉 EXTURE⁺는 EXTURE에 비하여 머신 트레이딩에 보다 더 친화적인 시스템이라고 할 수 있다.

## HFT 혹은 HST

EXTURE⁺를 추진할 당시만 해도 머신 트레이딩의 큰 흐름은 고빈도매매(High Frequency Trading, 이하 HFT)였다. 그렇지만 HFT에 대한 국제적인 규제가 강화되고 한국거래소도 알고리즘 트레이딩 관리 방안을 통해 과다호가 분담금 및 과다호가 규제를 제도화했다. 여기에 파생상품시장 건전화 정책과 경제 불황으로 유동성 부족장세가 이어지고 있다. 이로 인해 EXTURE⁺의 기반이라고 할 수 있는 HFT가 크게 위축된 상황이다.

그러면 EXTURE⁺의 미래는 불투명할까? 어떤 전략이라고 하더라도 HFT를 정의할 때 따르는 공통된 개념이 있다. 바로 로우 레이턴시다. 로우 레시턴시를 다른 말로 하면 고속(High Speed)이다. HFT를 대신하여 등장한 빅데이터 트레이딩은 중빈도(Mid Frequency) 지만 반드시 '속도'를 요구한다. 대용량 데이터를 빠른 시간에 처리하면서 기회를 포착해 이익을 실현해야 하기 때문이다. 이런 머신 트레이딩은 HFT가 아니라 HST(High Speed Trading)라고 부를 수 있다. 한마디로 EXTURE⁺의 등장으로 머신 트레이딩이 한 단계 변화한 HST에 대응하는 시스템에 더욱 유용해지는 셈이다.

> "EXTURE⁺의 등장은 지금보다 더 빠른 매매 처리 시스템 수요가 늘었다는 뜻이다. 자바는 이를 구현하는 기반이 된다."

## 로우 레이턴시와 JAVA

JAVA와 C/C++를 비교할 때 JAVA의 장점은 무엇일까. JAVA는 C/C++에 비해 개발자가 많고 커뮤니티가 발전해 있어서 재사용이 가능한 라이브러리나 프레임워크를 쉽게 구할 수 있다. 또한 C/C++에 비해 생산성이 뛰어나기 때문에 시장 환경 변화에 빨리 대응할 수 있다.

그렇지만 HFT시스템을 개발하고자 할 때 JAVA보다는 C/C++를 선호한다. 그 이유는 'JAVA는 느리다'는 편견이 작용하기 때문이다. 그러나 이때 '느리다'는 표현은 상대적이다. 1초를 놓고 비교할 때 1밀리세컨드는 1,000배 빠르다. HFT시스템에서 기준 시간은 마이크로세컨드다. 밀리세컨드보다 최소 100배 이상 빠른 속도를 요구한다.

만약 머신트레이딩에 요구되는 처리속도가


밀리세컨드라면 JAVA는 아주 훌륭한 개발언어다. 실제로 오픈 소스인 복합 이벤트 처리(Complex Event Processing)엔진 '에스퍼'(Esper)와 FIX(Financial eXchange Protocol)엔진인 '퀵FIX/J'(QuickFIX/J)를 적용한 알고리즘 트레이딩 시스템이 JAVA를 이용한 대표적인 사례다.

C/C++개발자들이 로우 레이턴시 환경을 구축하기 위하여 기본으로 도입하는 것이 TOE(TCP/IP Offload Engine)를 지원하는 네트워크카드(RNIC)다. 이것을 기반으로 하여 인메모리 프로세싱(In-Memory Processing)을 위한 기술과 멀티 스레드(Multi-Thread) 기반 기술을 기초로 애플리케이션을 개발한다.

그렇지만 마이크로세컨드를 다투는 시스템에서 요구하는 메모리 기술과 스레드 기술은 다르다. x86이라는 하드웨어 환경, 특히 멀티코어/멀티CPU 환경과 리눅스와 같은 OS에 대한 이해를 전제로 한다. CPU 캐시와 OS의 문맥전환(context switch) 및 인터럽트(interrupt)와 같은 요소들을 고려해야 한다. 여기에 멀티코어 환경이 보편화하면서 코어 투 코어(Core-To-Core) 간의 IPC 메시징이 중심이 되어야 한다.

JAVA 애플리케이션은 다른 애플리케이션과 다르게 작동한다. 즉 일반적인 애플리케이션은 '운영체제 – 애플리케이션'이지만 JAVA의 경우 '운영체제 – JAVA 가상머신(JVM) – 애플리케이션'과 같은 실행구조를 갖는다. 즉 JVM의 성능과 기능이 애플리케이션 성능을 좌지우지하는 것이다. JVM은 기본적으로 멀티스레드를 지원하는 환경이다. 이 때문에 개발 언어 차원에서 스레드를 보다 쉽게 생성하고 간단하게 동기화할 수 있는 기능을 제공한다. 다만 멀티 코어/멀티 프로세서 컴퓨터가 늘어나면서 다중 스레드를 사용한 프로그램에서 예기치 못한 데드록 상황이 종종 발생하고 이를 해결하기 위하여 JDK5 이후 도입한 JAVA메모리모델(JMM)이 커다란 전환점이 되었다.

스레드로 프로그램을 개발할 때 고려해야 하는 원자성(Atomicity), 가시성(Visibility) 및 실행순서(Ordering)를 지원하는 환경을 제공하기 때문이다. JVM은 환경일 뿐이다. 즉 실제로 HST에 JAVA를 채택하려면 JVM의 발전뿐 아니라 거쳐야 할 관문이 많이 있다. 먼저 JMM이 록(Lock)과 관련한 다양한 기능을 제공하지만 레이턴시 측면에서 Lock은 고비용이라 Lock를 사용하지 않는 알고리즘 – 다중 스레드 처리를 위한 록 프리 알고리즘(Lock-Free Algorithm)이나 캐시 프렌들리 알고리즘(Cache Friendly Algorithm)을 구현하여야 한다.

또 하나 중요한 것은 가비지컬렉션(Garbage Collection)과 관련한 부분이다. JAVA에서 가비지(Garbage)를 줄인다는 말은 CPU캐시를 쓸데없는 가비지로 채우지 않는다는 말과 같다. CPU캐시를 잘 사용하려면 캐시 미스를 줄여야 한다.

디스트럽터(Distruptor)를 개발한 LMAX의 마틴 톰슨은 이상과 같은 접근법을 미캐니컬 심퍼시(Mechanical Sympathy)라고 부른다. 세계 최대 자동차 경주대회인 포뮬러 F1에서 3회 우승을 차지한 재키 스튜어트가 한 말인 "베스트 드라이버는 기계가 어떻게 작동하는지 충분히 이해해야만 하모니를 이룰 수 있다"에서 영감을 얻은 용어다.

C/C++든 JAVA든 로우 레이턴시 환경을 구축하고 속도를 얻고자 하면 하드웨어(특히, CPU)를 깊이 이해하고 이를 기반으로 한 애플리케이션을 개발해야 한다. 이 전제 조건이 충족돼야 비로소 원하는 성능을 얻을 수 있다는 의미이다.

## JAVA로 고속트레이딩을 구현하다

2011년 후반 JAVA를 HST/HFT에 적용한 사례가 처음 등장한다. 대표적인 사례는 외환거래를 중개하는 영국 LMAX 거래소다. FX와 차액거래(CFD)를 위한 매매체결시스템은 광학 전송계(MTF) 지연을 기준으로 500마이크로세컨드의 성능을 제공한다.

이러한 LMAX의 매매체결 시스템을 개발한 사람은 앞서 소개한 마틴 톰슨으로 JAVA를 기반으로 디스트럽터를 개발하여 적용했다. 디스트럽터는 인터-스레드 커뮤니케이션 프레임워크(Inter-Thread Communication Framework)다.

디스트럽터의 핵심은 백만 개의 슬롯을 가진 한 개의 거대한 원형 큐(Ring buffer Queue)다. 각 태스크는 자신이 슬롯의 어디까지 처리했는지 나타내는 플래그 숫자를 가지며, 자신의 앞에 있는 태스크의 숫자를 넘어서 수행할 수는 없다.

이 숫자 하나만을 사용해서 동기화가 이루어지기 때문에 동기화 오버헤드가 매우 적고, 하나의 거대한 원형 큐를 사용하기 때문에 기존 방식에서처럼 큐의 공간 부족으로 블록되는 경우가 발생하지 않는다. 또 록의 사용도 배제하여 높은 성능과 빠른 속도를 구현했다.

또 다른 사례는 오픈 소스 프로젝트인 오픈 HFT(OpenHFT)다. 피터 로레이가 주도하는 프로젝트로 퍼시스티드 로우 레이턴시 메시징(Persisted low latency messaging)을 위한 JAVA-크로니클, JAVA 스레드 어피니티 라이브러리(Thread Affinity library)인 JAVA-스레드-어피니티 및 JAVA 사용 오프 힙 스토리지의 거대한 모음(Huge Collect-ions for Java Using Efficient Off Heap Storage)을 위한 거대 모음(Huge Collect-ions)으로 구성되어 있다.

"성능에 가장 큰 영향을 주는 부분은 낮은 수준의 언어로 개발하고 나머지는 생산성이 높은 언어로 개발하자"는 정신에 따라 OpenHFT 프로젝트는 로 레벨(Low Level) JAVA와 관련한 라이브러리를 제공하고 있다.

JAVA 애플리케이션을 실행하는 JAVA 가상머신인 JVM도 초고속에 적합한 방향으로 진화하고 있다. 오라클은 로우 레이턴시 JAVA라는 개념으로 가비지 컬렉션을 개선한 JVM을 내놓았다.

또 아줄 시스템스(Azul Systems)는 로우 레이턴시 환경에 특화한 JVM인 징(Zing)을 내놓았다. 실시간처리, 고속처리, 대용량처리 등에 대한 요구에 따라 JAVA는 로우 레이턴시로 변화하고 있음을 보여주는 것이다.

IT와 관련한 하드웨어나 소프트웨어는 끊임없이 진화한다. C/C++처럼 JAVA도 산업적 필요에 따라 계속 진화하고 있다. 물론 JAVA가 C/C++를 밀어내고 HST의 표준적인 환경이 될 수 있을지는 아직 알 수 없다. 하지만 중요한 것은 기술이 아니라 기술을 다루는 사람이다. 그리고 이 점이 HST에서 JAVA의 미래가 나쁘지 않은 이유이다.

> "실시간처리, 고속처리, 대용량처리 등에 대한 요구에 따라 자바는 로우 레이턴시로 변화하고 있다."